VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

FORENZNÍ ANALÝZA MALWARE

FORENSIC MALWARE ANALYSIS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. BENJAMIN KRÁL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. PAVEL OČENÁŠEK, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Tato diplomová práce popisuje metody a postupy používané při forenzní analýze malware, včetně metod statické i dynamické analýzy malware. S využitím popisovaných metod je poté navrhnut nástroj určený k užívání bezpečnostními týmy CSIRT, jež vyšetřovateli bezpečnostního incidentu umožní rychle analyzovat a rozhodnout roli vzorku malware s nímž se setká při šetření bezpečnostního incidentu. Tento nástroj je v rámci práce podrobně popsán v odborném technickém návrhu založeném na specifických požadavcích bezpečnostních týmů CSIRT specifikovaných taktéž v obsahu práce. Na základě tohoto návrhu je implementován nástroj ForensIRT, jež je následně otestován analýzou vzorku malware Cridex. Konečně výsledky této analýzy jsou porovnány s výsledky ostatních srovnatelných nástrojů určených k forenzní analýze malware.

Abstract

This master's thesis describes methodologies used in malware forensic analysis including methods used in static and dynamic analysis. Based on those methods a tool intended to be used by Computer Security Incident Response Teams (CSIRT) is designed to allow fast analysis and decisions regarding malware samples in security incident investigations. The design of this tool is thorougly described in the work along with the tool's requirements on which the tool design is based on. Based on the design a ForensIRT tool is implemented and then used to analyze a malware sample Cridex to demonstrate its capabilities. Finally the analysis results are compared to those of other comparable available malware forensics tools.

Klíčová slova

Computer Security Incident Response Team, CSIRT, CERT, Malware, Forenzní analýza, Statická analýza, Dynamická analýza, Nástroj

Keywords

Computer Security Incident Response Team, CSIRT, CERT, Malware, Forensic analysis, Static analysis, Dynamic analysis, Tool

Citace

KRÁL, Benjamin. *Forenzní analýza malware*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Pavel Očenášek, Ph.D.

Forenzní analýza malware

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Pavla Očenáška, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Benjamin Král
18. Května 2018

Poděkování

Děkuji svému vedoucímu Ing. Pavlovi Očenáškovi Ph.D. za jeho odbornou pomoc a dohled při psaní této diplomové práce. Dále bych chtěl poděkovat bezpečnostnímu týmu Masarykovy Univerzity CSIRT-MU a Norskému národnímu bezpečnostnímu týmu NorCERT, jež mi pomohli porozumět a vyzkoušet si každodenní činnosti spojené s řešením bezpečnostních incidentů.

Obsah

Kapitola 1

Úvod

V dnešním světě jsou informační technologie nedílnou součástí života každého z nás. Počítače, mobilní telefony, tiskárny, automobily, to vše je dnes vybaveno inteligentními čipy, které dokážou vykonávat s vysokou přesností složité výpočetní úkony. Na spoustu takovýchto zařízení lze nainstalovat také vlastní programy, který umožní uživateli rozšířit schopnosti daného zařízení. Ovšem ne vždy daný program vykonává činnost, jakou by uživatel od dané aplikace očekával. Tato chytrá zařízení jsou navíc stále výkonnější a přináší stále novější funkce, ve kterých se uživatelé často neorientují a nerozumí bezpečnostním rizikům s využíváním informačních technologií spojenými. Tvůrci těchto škodlivých programů, obecně nazývanými malware, zapojují různé klamavé techniky a bezpečnostní mezery v již existujícím programovém vybavení přístroje, aby získali přístup a moc nad zařízením a popřípadě zajistili šíření malware na další zařízení.

Při pohledu na kybernetické bezpečnostní incidenty za poslední desetiletí je zřejmé, že malware hraje důležitou či přímo hlavní roli v mnoha z těch nejznámějších a potenciálně nejvíce poškozujících incidentech. Zářný případ je třeba malware nazývaný Stuxnet, poprvé odhalený v roce 2010, o kterém se obecně předpokládá, že jeho cílem bylo poškodit přístroje v zařízení k obohacování jaderného paliva. Yannakogeorgos a Tikk ve svém článku [24] nevylučují možnost, že tento incident byl iniciován jiným státem a že tento vysoce sofistikovaný malware byl navržen a vytvořen na míru tak, aby napadl cílené zařízení patřící jinému státu a dle názoru některých odborníků by proto mohla být tato akce považována za válečný akt. [21, 24]

Ovšem tato tvrzení jsou velmi obtížně dokazatelná, protože takto sofistikovaný malware je často tak komplexní a rozsáhlý, že jeho analýza trvá řádově několik let. Autoři malware navíc často zapojují různé techniky s cílem ztížit vyšetřování a analýzu jejich malware, čímž dále ztěžují již tak obtížné podmínky forenzní analýzy malware.

Ovšem problematika útoků malware se netýká pouze státních zařízení či kritické infrastruktury. Malware je velmi nebezpečná hrozba i pro běžné koncové uživatele, respektive pro jejich zařízení. Podle statistik pro rok 2016 vydaných bezpečnostní firmou Kaspersky [5] bylo 31.9% uživatelských počítačů za rok 2016 napadeno z internetu alespoň jedním vzorkem malware a jen za rok 2016 tato bezpečnostní firma zabránila více než tři čtvrtě miliardě útoků na uživatelské systémy. A to se jedná pouze o jednu z mnoha bezpečnostních firem, takže celková čísla zabráněných ale i nezabráněných útoků v tomto roce jsou mnohem vyšší. [5]

Také z tohoto důvodu vznikají bezpečnostní týmy CSIRT (z angl. Computer Security Incident Response Team) [14, 13], které zajišťují řešení incidentů pro dílčí části sítě Internet, které jsou pod jejich bezpečnostní správou, resp. tyto týmy mají autoritu řešit incidenty

dotýkající se této části sítě. Tyto bezpečnostní týmy se mimo jiné setkávají také s incidenty jejichž součástí je nějaký vzorek malware. Protože však zdroje takových týmů mohou být často značně omezené, nemohou vždy provádět hloubkovou analýzu vzorků malware, či vůbec provádět nějakou analýzu, přestože by toto bylo na místě.

Tato diplomová práce rozebírá nastíněný problém a navrhuje řešení v podobě nástroje využitelného bezpečnostními týmy k základní forenzní analýze zjištěného vzorku malware. Nejdříve v kapitole 2 nahlédneme do problematiky forenzní analýzy malware, zaměříme se existující techniky a postupy forenzní analýzy malware a do detailů rozebereme způsoby získávání některých vybraných metadat ze vzorků malware. V následující kapitole prozkoumáme existující nástroje jež umožňují sběr těchto dat a provedeme jejich zevrubnou analýzu. Následně v kapitole 5 na základě nabytých znalostí navrhneme a implementujeme nástroj, který je vhodný k použití bezpečnostními týmy k rychlé a přehledné analýze tak, aby i týmy s omezenými zdroji mohly provést bez většího úsilí alespoň základní statickou analýzu. Konečně navržený systém implementujeme a za použití vhodného vzorku malware provedeme pomocí tohoto nástroje forenzní analýzu zpracovanou ve formě forenzního posudku.

Kapitola 2

Techniky a postupy forenzní analýzy malware

Abychom se mohli zabývat forenzní analýzou malware a navrhnout nástroj který by tuto činnost usnadňoval, musíme nejdříve detailně nastudovat jak forenzní analýza malware funguje, tedy zjistit jaké techniky a postupy využívají bezpečnostní týmy CSIRT při forenzní analýze vzorků malware, s nimiž se setkávají při řešení bezpečnostních incidentů. Proto v této kapitole identifikujeme nejzajímavější informace a metadata jež můžeme získat ze vzorku malware a detailně prozkoumáme jakým způsobem můžeme tato data z malware získat.

2.1 Zájmové typy malware

Bezpečnostní týmy CSIRT jež chrání běžnou počítačovou infrastrukturu sítě v jejich bezpečnostní správě či autoritě se nejčastěji setkávají s běžnými vzorky malware, nikoliv s vysoce sofistikovanými novými vzorky malware, k nimž je potřeba provést zevrubnou analýzu. V roce 2016 bezpečnostní tým CSIRT.cz, národní bezpečnostní tým pro Českou Republiku, zaznamenal 5109 unikátních vzorků malware avšak pouze 41 vzorků vyhodnotili jako zajímavé vzorky do takové míry, že bylo žádané je předat k dalšímu zkoumání společnosti Avast. [4]

Jak poukazují Wiik a spol. [22], CSIRT týmy bývají často velmi vytíženy a nemají proto ani dostatek prostředků řešit incidenty s nízkou prioritou, což může vést k zanedbávání a nekvalitnímu řešení těchto incidentů. Z tohoto důvodu se bezpečnostní týmy mohou věnovat forenzní analýze malware jen na základní úrovni.

Z těchto důvodů a z mé vlastní zkušenosti při spolupráci s CSIRT-MU, bezpečnostním týmem Masarykovy Univerzity v Brně, vyplývá, že nejčastěji se tyto bezpečnostní týmy zabývají forenzní analýzou běžných vzorků malware, tedy nepříliš sofistikovaných vzorků, u nichž je důvod analýzu provést (např. je to malware dříve antivirovým softwarem neviděný, což podle výroční zprávy CSIRT.cz pro rok 2016 [4] bylo přibližně 44% všech unikátních zjištěných vzorků). Typickou ukázkou takového malware je například malware třídy "downloader", neboli program jehož výlučný účel je stažení dalšího škodlivého kódu na infikovaný stroj (popř. podobné třídy malware jako například tzv. "dropper"či "unpacker", které se liší například tím, že škodlivý kód který umístí na infikovaný stroj nestahují ze sítě, ale již jej v nějaké podobě obsahují ve svém kódu). Další zájmová třída malware je například tzv. "botnet", což představuje malware jež se nainstaluje na infikovaný systém a poté

poslouchá na určitém komunikačním kanálu a vykonává příkazy na tomto kanálu odposlechnuté. V neposlední řadě bezpečnostní týmy řeší incidenty s malware třídy "backdoor", což lze z angličtiny volně přeložit jako zadní vrátka. Malware této třídy je podobný třídě botnet avšak neposlouchá na komunikačním kanálu společném pro všechny další infikované stroje, ale dává autorovi tohoto malware skrytý přístup k systému kde je tento malware nainstalovaný individuálně. [17]

2.2 Řetězce v binárních souborech

Technicky pravděpodobně nejjednodušší ale zároveň často jednou z nejužitečnějších technik je určitě extrakce textových řetězců z binárních souborů malware. Tato technika, jak název napovídá, se snaží najít smysluplné textové řetězce v binárních souborech, tedy posloupnosti bajtů s hodnotami v rozsahu tisknutelných znaků zakončených bajtem s hodnotou nula.

Je zřejmé že tato technika je v principu triviální dolování dat z binárních souborů, ovšem často může být značně účinná a bývá zdrojem vysokého počtu artefaktů, jež některé mohou být pro forenzní vyšetřování kritické. Mezi takové kritické artefakty získané touto metodou můžou patřit různé řetězce jako IP adresy a doménová jména, se kterými může malware komunikovat, příkazy kterými malware komunikuje přes internet (často je jako komunikační kanál volen například textový komunikační protokol IRC [12], jenž je pomocí jeho příkazů snadno rozpoznatelný) či třeba případné dešifrovací klíče pro zašifrované úseky kódu a dat. [17, 18]

Nicméně ačkoliv se může zdát tato metoda jako velmi jednoduchá a přímočará, je třeba pamatovat na několik technických komplikací a problémů s touto metodou spojených. Značnou komplikací může být například kódování řetězců v binárních souborech. Tento problém demonstruji na následujícím příkladu, kdy řetězec ve vzorku malware reprezentuje potenciální příkaz protokolu IRC k připojení do kanálu `commchan`:

```
J  O  I  N  _  #  c  o  m  m  c  h  a  n
4f 4a 4e 49 23 20 6f 63 6d 6d 68 63 6e 61 000a
```

Je zřejmé že v kódování ASCII kde každý znak je reprezentován 7 bity (nejvýznamnější bit je nulový) a řetězec je zakončen bajtem s hodnotou nula je velmi odlišný např. od řetězce v kódování UTF-16, jež má symboly seskupené do dvojic, symboly ve dvojici jsou prohozeny a navíc každý symbol je reprezentován dvěma bajty, což v případě použití pouze standardních tisknutelných znaků znakové sady ASCII znamená, že každý jednotlivý symbol se skládá z jednoho nenulového a druhého nulového bajtu:

```
    JO       IN       _#       co       mm       ch       an
fffe 4a004f00 49004e00 20002300 63006f00 6d006d00 63006800 61006e00 0a00
```

V případě že bychom použili algoritmus pro nalezení řetězců ve standardním kódování ASCII na binární soubor obsahující řetězce v kódování UTF-16, výstup algoritmu by byl jen opravdu velmi těžce interpretovatelný, resp. by řetězce v kódování UTF-16 nebyly nalezeny vůbec. Tento problém je navíc znásoben faktem, že formátů textových řetězců jsou desítky a řetězce mohou být potenciálně v jakékoliv z těchto formátů.

Další problém v této technice vychází ze skutečnosti, že binární soubory mohou přirozeně obsahovat náhodná uskupení tisknutelných symbolů zakončených bajtem s hodnotou nula. Výstupem této techniky je proto obrovské množství artefaktů, z nichž jen velmi malá část je pro forenzní vyšetřování užitečná. Je proto na místě zapojit různé techniky filtrování získaných artefaktů, aby byly vybrány jen ty nejzajímavější a forenzní vyšetřovatel nebyl

zahlcen stovkami nesmyslných řetězců. Mezi tyto techniky výběru vhodných řetězců může patřit například jednoduché omezení získaných řetězců na ty, jejichž délka je nejméně daný počet symbolů. Dále se můžeme dívat například na entropii, počet alfanumerických resp. speciálních znaků apod.

2.3 Metadata v záhlaví Portable Executable

Binární spustitelné soubory pro platformu Windows jsou v dnešní době v drtivé většině ve formátu PE [9] (z angl. Portable Executable), což je datová struktura jež definuje přesný formát, jaký mají spustitelné aplikace pro operační systém Windows mít. Tato struktura kromě samotného kódu aplikace (v podobě zkompilovaného strojového kódu) a dat aplikace definuje také záhlaví této datové struktury, kde lze najít podrobné informace o daném programu. V rámci forenzní analýzy malware se rozlišuje statická a dynamická analýza, kde statická analýza je analýza statických dat daného malware. A právě tato hlavička, mimo samotného kódu programu, je jeden z ústředních zdrojů dat ve statické analýze, především díky skutečnosti, že tato hlavička je zpravidla dostupná ihned na počátku vyšetřování a může poskytnout první náhled na parametry a možné schopnosti analyzovaného malware. Z tohoto důvodu se nyní detailně podívejme na tuto datovou strukturu se zaměřením na údaje, které nás v rámci forenzní analýzy malware nejvíce zajímají. [20, 9, 2]

2.3.1 Záhlaví DOS

Každý PE spustitelný soubor začíná záhlavím DOS (angl. DOS Header) a oddílem DOS Stub. Tento oddíl může mimo jiné obsahovat celý program pro DOS[1], z toho také vyplývá jméno tohoto oddílu. Běžně bývá v tomto oddílu uložen jen drobný program, který v případě spuštění programu určeného pro moderní operační systémy vypíše řetězec "*This program cannot be run in DOS mode.*"nebo jemu obdobný. Tato část však není až tak důležitá pro forenzní analýzu, jako je důležitý první a poslední blok v záhlaví DOS. První blok o velikosti 2 bajtů je význačný svým obsahem - obsahuje znaky `MZ`, což je identifikátor že se jedná o spustitelný soubor v tomto formátu a nazývá se `e_magic`. Poslední blok záhlaví s názvem `e_ifanew` o velikosti 4 bajty (nacházející se v souboru na adrese resp. odsazení `3C`) určuje adresu odsazení dalšího významného oddílu - záhlaví NT. [9]

První blok `e_magic` je z hlediska forenzní analýzy obzvláště zajímavý především protože je vždy v každém spustitelném souboru na prvním místě a má konstantní hodnotu. Proto pokud vyšetřovatel ví, že se má jednat o spustitelný soubor (např. se jedná o vzorek stažený jiným malware třídy "downloader"), avšak na začátku souboru nejsou znaky `MZ`, pak je pravděpodobné, že daný soubor je zašifrovaný. Navíc pokud se jedná o jednoduchou šifru jako např. monoalfabetická šifra (a jiné jednoduché šifry náchylné k rozluštění základní kryptoanalýzou), může toto spolu s řetězcem informujícím o nemožnosti spustit tento program v módu DOS pomoci zjistit šifrovací klíč a dešifrovat daný program, protože hodnoty obou těchto bloků dat jsou známy. [17, 9, 2]

Poslední blok záhlaví DOS je důležitý především z praktického hlediska - obsahuje totiž adresu začátku záhlaví NT, tedy míří za oddíl DOS Stub, který je v dnešních operačních systémech prakticky zbytečný. [9]

[1] DOS - angl. Dos Operating System, v češtině Diskový Operační systém. Jedná se o rodinu operačních systémů z nichž nejznámější je MS-DOS.

2.3.2 Záhlaví NT

Záhlaví NT znovu začíná blokem s konstantní velikostí, jako tomu bylo i u záhlaví DOS, ovšem tentokrát se symboly `PE` (a následně dvěma nulovými bajty) a názvem `signature`. Ovšem NT záhlaví obsahuje také několik mnohem významnějších polí pro analýzu malware. [9]

V poli `machine` následujícím ihned za polem `signature` je hodnota identifikující pro jakou architekturu je tento spustitelný soubor určen (nejčastěji tedy bude toto pole nabývat hodnot i386 a amd64). Tato informace je význačná především pro další analýzu daného vzorku, protože od této skutečnosti se odvíjí mnoho formalismů v samotném strojovém kódu aplikace. Hned 4 bajty za tímto polem následuje další poměrně významné pole a to `timedatestamp`, jež obsahuje datum a čas kdy byl tento soubor zkompilován, zakódované v podobě časového razítka reprezentující počet sekund jež uběhlo od data 1. 1. 1970. Toto je významné v případě že datum zde uvedené je očividně nesmyslné. Například pokud míří do budoucnosti, nebo naopak příliš daleko do minulosti. Z technického hlediska totiž běžný program nemá důvod toto datum měnit, proto je určitě jeho nesmyslný obsah znamením pravděpodobné modifikace způsobené nějakým potenciálně škodlivým záměrem autora.

Součástí záhlaví NT je také nepovinný oddíl (v angl. Optional Header) jehož délka je určena polem s názvem `sizeofoptionalheader`. V tomto oddíle můžeme nalézt další významná pole. Například ihned první pole `magic` zdrží informaci o tom, jestli je tento spustitelný soubor určen pro 32 nebo 64 bitové architektury, což je významné z podobných důvodů jako hodnoty v poli `machine`. Další velmi významná hodnota je hodnota pole `addressofentrypoint`, která jak již název napovídá určuje samotný začátek programu, resp. místo odkud má začít počítač začít vyhodnocovat příkazy strojového kódu. Konečně jedno z dalších významných polí je `subsystem`, které určuje jaký typ aplikace tento soubor identifikuje (nejčastěji pro malware pro moderní operační systém Windows bude nabývat hodnot `0003h` což značí konzolovou aplikaci, nebo hodnotu `0002h` pro Windows GUI, neboli aplikaci s grafickým prostředím). [9]

Velmi významný z hlediska forenzní analýzy malware je také pole `imagebase`, které určuje na jaké adrese v paměti má být dle tvůrce umístěn obraz programu. Výchozí hodnota je vždy `00400000h` a podobně jako u `timedatestamp` jiná než očekávaná hodnota může být znamení, že autor programu se snaží dělat něco nestandardního a potenciálně škodlivého. [9]

2.3.3 Záhlaví Sekcí

Za záhlavím NT následuje záhlaví sekcí (angl. Section Headers), které je také velmi důležité. Určuje totiž jaké sekce se v tomto programu vyskytují. Dnes se setkáváme nejčastěji s osmi následujícími sekcemi:

- `.text` obsahuje samotný strojový kód aplikace tak, jak je zkompilován ze zdrojového kódu.
- `.bss` obsahuje neinicializovaná data, tedy neinicializované statické a dynamické proměnné.
- `.rsrc` obsahuje všechny zdroje (angl. resources), jako například obrázky, zvuky, ale někdy i třeba řetězce v nestandardním formátu/kódování a další podobné objekty které může aplikace obsahovat.

- `.data` obsahuje inicializovaná data, tedy lokální a globální proměnné a mimo jiné i řetězce, jejichž hodnota je inicializována již při kompilaci.
- `.rdata` obsahuje adresář s debugovacími informacemi a někdy také data určená pouze pro čtení, jako třeba konstanty, nebo řetězce.
- `.edata` obsahuje popis funkcí a metod jež tento soubor exportuje. Tato sekce je významná především pro knihovny .dll, protože soubory .exe většinou funkce neexportují.
- `.idata` obsahuje popis funkcí a metod jež tento soubor importuje.
- `.reloc` obsahuje relokační tabulku - seznam adres v kódu, které je třeba přepočítat v případě že systém nemůže našíst program na adresu, kam by měl být umístěn.

Záhlaví sekcí a posléze samotné sekce jsou pro forenzní analýzu malware základní a velmi důležitý zdroj informací. Právě sekce `.idata` totiž mimo jiné obsahuje odkaz na importní složku (angl. import directory) jež obsahuje seznam všech importovaných knihoven a funkcí, z jíž může v první fázi analýzy forenzní vyšetřovatel vyčíst předpokládané možnosti a schopnosti aplikace. Proto si vyšetřovatel může udělat ihned obrázek o tom, jestli má konkrétní vzorek schopnost připojit se k internetu, jestli umí pracovat se soubory, semafory či jinými zdroji a na základě těchto informací může hledat další artefakty (např. jestliže aplikace má schopnost připojit se k internetu, pak se zřejmě někde v datech bude nacházet identifikátor cíle připojení, jako např. IP adresa nebo doménové jméno). [9]

Ovšem ne vždy tento importní adresář obsahuje seznam všech importovaných databází. Jestliže daný program využívá API poskytnuté systémem k linkování knihoven až za běhu aplikace (angl. runtime linking), pak nemusí uvádět importované knihovny v této složce a jednoduše si tímto způsobem nalinkuje knihovny až ve chvíli kdy jsou potřeba. Ovšem protože k tomuto bude využívat API poskytnuté systémem, tato skutečnost se do importního adresáře projeví minimálně prezencí těchto funkcí - typicky jsou to funkce `LoadLibrary`, `GetProcAddr`, `LdrGetProcAddr` a `LdrLoadLibrary`. Tato možnost je navíc hojně využívána programy jež jsou zabalené, zkomprimované či zašifrované a jejich kód je zavaděč jenž rozbalí resp. rozšifruje kód samotné aplikace a ta již načte požadované knihovny za běhu tímto způsobem. Tento přístup je často využíván tvůrci malware ke skrytí funkčnosti programu, ovšem občas je možné se s ním setkat i u legitimních aplikací. [17, 9]

Další významná informace nacházející se v záhlaví sekcí z hlediska forenzní analýzy malware je avizovaná velikost jednotlivých sekcí. U každé sekce jsou vypsány 2 informace týkající se jejich velikosti a to surová velikost `RawSize` a virtuální velikost `VirtualSize`. Surová velikost vyznačuje jak velká je tato sekce v daném binárním souboru zatímco virtuální velikost informuje systém, kolik místa má pro daný oddíl alokovat v paměti. Pokud tedy například je virtuální velikost mnohem větší než surová velikost, pak zřejmě daná sekce bude za běhu naplněna daty. Typicky malware jež rozbalí svůj kód má velmi malou surovou velikost sekce `.text` a velmi velkou virtuální velikost této sekce, čímž zajistí alokování velkého prostoru do kterého načte samotný kód aplikace například ze sekce `.rsrc` (která tím pádem bude znatelně velká, aby mohla obsáhnout právě zabalený resp. zašifrovaný kód programu). Typicky toto chování nen výlučné pro malware, ovšem tvůrci malware tyto techniky hojně využívají a vyšetřovatel incidentu by určitě měl při analýze takového vzorku věnovat zvláštní pozornost této skutečnosti, protože toto jasně upozorňuje na fakt, že kód programu je autorem záměrně skryt, zabalen či zašifrován a je třeba prověřit proč tomu tak je. [17, 9]

2.4 Statická analýza programového kódu

Konečně nejdetailnější, ale i nejobtížnější a časově nejnáročnější technikou je analýza samotného kódu programu. Tato technika spočívá v dekompilaci (resp. překladu) strojového kódu aplikace do asembléru (resp. jiného vyššího programovacího jazyka či pseudokódu, ovšem zde je problém s rekonstrukcí kódu tak aby byla zachována jeho exaktnost a přehlednost, proto je většinou tato technika založena především na asembléru) a následné analýze kódu programu, odhadnutí jeho činnosti a získání dalších informací a dat z kódu programu.

Přestože tato metoda je nejefektivnější, je však také zajisté časově nejnáročnější, proto v prostředí analýzy incidentů týmy CSIRT zpravidla nepřichází v úvahu, protože jak již bylo zmíněno, týmy CSIRT mají velmi omezené možnosti a zdroje, zpravidla jsou zahlceny incidenty a zároveň se musí pokoušet vyvíjet a udržovat vlastní nástroje apod. [22], proto nemohou věnovat jednotlivým incidentům mnoho času a proto tuto časově náročnou metodu příliš nevyužívají a přiklánějí se spíše k automatickým či jednodušším (avšak méně informativním) metodám. Z tohoto důvodu, společně s důvodem omezeného prostoru v této práci, se podíváme pouze na vybrané nejběžnější a nejdůležitější artefakty jenž můžeme při statické analýze v kódu programu najít a to artefakty, jenž jsou milníky při forenzní analýze malware.

Jednu z věcí na které se můžeme při forenzní analýze malware dívat je volání systémových API funkcí. Konkrétně nás budou zajímat především přístupy ke zdrojům operačního systému, například k souborům. Při analýze malware se snažíme mimo jiné zjistit jaké změny provede s operačním systémem a programovým vybavením zařízení, na kterém je spuštěn. Proto hledáme například právě zápisy do souborů. Podle toho ke kterým souborům se vzorek snaží přistupovat můžeme zjistit zda se jedná o malware a popř. jaký účel daný malware má. Je zřejmé, že běžný program by neměl přistupovat do určitých lokací, například do složky se systémovými soubory atp. Navíc detekce míst kde je k souborům v kódu přistupováno je ve většině případů poměrně přímočará, pouze stačí vyhledat příkazy jimiž je možno zavolat funkci (typicky příkaz `CALL`) a zjistit z argumentu jakou funkci volá, resp. v opačném pořadí - najít všechny instance příkazu `CALL` které volají právě funkce pro manipulaci se soubory. Podobně můžeme také přistupovat k dalším systémovým zdrojům, jako například přístupy k síti (kde můžeme po nalezení funkce k přístupu k síti také z argumentů této funkce zjistit například k jakému vzdálenému zdroji se snaží přistupovat apod.). [19]

Další důležitý systémový zdroj významný pro forenzní vyšetřovatele jsou semafory (angl. mutex). Malware může semafory využívat několika způsoby. Především je však využívá ke komunikaci se sebou samým - například malware může být vytvořen tak, aby před infekcí daného stroje se podíval zda je přítomný a uzamknutý semafor s určitým názvem. Jestliže není, infikuje počítač a tento semafor vytvoří a uzamkne. V případě že by došlo k opětovné infekci, malware ihned zjistí, že daný systém je již infikovaný a nebude se znovu pokoušet infikovat daný systém. Proto je pro forenzní vyšetřovatele důležité prověřit i tuto možnost. Více o semaforech je probráno v kapitole 2.7.

Kromě sledování přístupu k systémovým zdrojům můžeme v samotném kódu programu hledat i řadu dalších věcí, které mohou poskytnout indície ke skutečnosti, že daný vzorek je skutečně malware. Autoři malware se snaží zakrývat co skutečně dělá malware, ať již k znesnadnění práce forenzním vyšetřovatelům, či k znesnadnění detekce daného vzorku malware antivirovými programy. Právě proto autoři malware často zapojují šifrovací či komprimační algoritmy a proto také forenzní vyšetřovatel může hledat právě přítomnost těchto algoritmů. V praxi autoři malware většinou nevyužívají běžně dostupné knihovny

pro šifrování pomocí běžných moderních algoritmů, ale spíše se přiklánějí k psaní vlastních jednoduchých šifrovacích metod. Typicky jsou tyto metody poměrně krátký úsek kódu, jenž obsahuje smyčku procházející řetězec a na každý bajt aplikuje nějakou binární operaci (ve smyslu operace nad dvěma operandy - zdrojového bajtu a bajtu nějakého klíče, nikoliv ve smyslu operace nad binárními daty), typicky to bývá operátor XOR. [3]

Proto je možné vyhledávat právě tyto smyčky, či podobné kritické a unikátní krátké úseky kódu. K tomu přispívá také skutečnost, že autoři malware často takový kód opětovně užívají v budoucích vzorcích (byť třeba v mírně pozměněné podobě) a proto také již známý úsek škodlivého kódu může pomoci k budoucí detekci nových škodlivých vzorků malware. Také na této metodologii může být založena detekce vzorků malware pomocí signatur, tedy typická metoda moderních antivirových programů. Takové antivirové programy poté mohou obsahovat databázi signatur právě takových šifrovacích smyček a dalších unikátních úseků kódu a jestliže jsou tyto ve skenovaném vzorku zjištěny, je považován za malware. Avšak u vyhledávání signatur krátkých úseků kódu v testovaných vzorcích v této modelové situaci nástroje pro použití týmy CSIRT je problematická, převážně z důvodu že je třeba mít k dispozici velmi rozsáhlou databázi signatur, jež je třeba získat vhodným způsobem, což je rozsáhlý problém sám o sobě. [3, 25]

Jednou z dalších věcí, jež je možno v kódu sledovat, je entropie částí vzorku. Jak vyplývá ze základní teorie komprese dat [8], zkomprimovaná data mají typicky větší entropii než původní nešifrovaná data. Toho také můžeme využít k určení pravděpodobnosti zda například určitý oddíl testovaného vzorku má nezvykle vysokou entropii a tedy že může obsahovat komprimovaná či šifrovaná data. Jak popisují Lyda a Hamrock [7], během testů zjistili že šifrované či komprimované vzorky mají entropii v rozsahu významně vyšším než běžné vzorky malware bez komprimovaných či šifrovaných dat. Díky této skutečnosti můžeme vyšetřovateli poskytnout další informaci jež může poskytnout indícii ke skutečnosti, zda se jedná o vzorek malware. Výhodou této metody je navíc skutečnost, že vypočítat entropii úseku vzorku je poměrně snadné a není třeba příliš chápat přímo obsah daného úseku. [7]

2.5 Přístup k systémovým registrům

Kromě výše popsaných metod jež spadají do statické analýzy kódu, tedy zkoumání kódu vzorku bez jeho spuštění, můžeme také při analýze malware zapojit metody dynamické analýzy. Při dynamické analýze spustíme vzorek malware v bezpečném odděleném, speciálně předem připraveném prostředí. Toto prostředí může být pouze drobné prostředí, jež je striktně separováno od zbytku operačního systému, nazýváno odborně sandbox (česky pískoviště, avšak tento termín nebudu v práci překládat). Problém ovšem je, že autoři malware se samozřejmě snaží zabránit vyšetřovatelům v analýze jejich vzorku, proto zapojují různé techniky, které se snaží zjistit zda je daná instance vzorku malware spuštěna v takovém umělém prostředí (zpravidla různými metodami, např. specifickými instrukcemi procesoru, pokusem přístupu k existující či neexistující webové stránce atp.) a v případě že je tato skutečnost zjištěna, zastaví chod malware aby nebyla možná jeho analýza. Z tohoto důvodu se někteří vyšetřovatelé přiklání spíše ke spuštění vzorku ve virtuálním prostředí jež simuluje celý systém, včetně virtualizace hardware na němž virtuální systém běží. Ovšem i přes použití virtualizační technologie je stále možné zjistit skutečnost, že malware není spouštěn přímo na reálném hardware v běžně používaném systému (např. opět některými příkazy, pokusy k přístupu na internet, vyhledávání speciálních vyšetřovacích nástrojů jako např. IDAPro či Wireshark, nebo třeba vyhledání služeb souvisejících s během ve virtu-

álním prostředí jako je např. vmware-tools atp.). Tato oblast je žhavé téma pro výzkum v tomto oboru, především proto, že moderní pokročilé vzorky malware tyto metodologie zpravidla pokaždé zapojují (a proto například bylo možné překonat některé verze nedávného velmi nebezpečného ransomware WannaCry jež zjišťoval dostupnost určitých domén v síti internet a na tomto základě (ne)infikoval daný stroj). [1, 17]

Jestliže testovaný malware nezapojuje tyto techniky detekce simulovaného či virtuálního prostředí, nebo je možné je obejít, můžeme testovat malware pomocí dynamické analýzy, tedy spustit jej a pomocí různých metod a nástrojů sledovat jejich běh a chování a na základě zjištěných informací daný vzorek klasifikovat. Typicky metodologie sledování chování spadá do jedné ze dvou kategorií. První kategorie metod sledování chování vzorku se skládá z metod jež přímo sledují chování malware a sledují volané příkazy API, jejich parametry atd. Druhá kategorie metod se nedívá přímo na chování programu, protože toto chování může být velmi složité a orientace v něm může být obtížná, ale sleduje naopak změny v systému, jež jsou zavedeny od spuštění vzorku. Typicky se tedy zajistí snímek celého virtuálního prostředí, následně se spustí vzorek malware, nechá se běžet vhodně dlouhou dobu, ukončí se a opět se získá snímek virtuálního systému. Tyto dva snímky se poté mohou různými způsoby porovnat a jejich rozdíly upozorní na změny zavedené v době běhu vzorku. Obě kategorie mají své výhody i nevýhody, první kategorie je typicky velmi složitá a bohatá na informace (které ovšem mohou vést i k zahlcení, zakrývání skutečné činnosti apod.), vhodná především na následnou automatickou analýzu, klasifikaci a predikci na základě chování malware. Druhá kategorie je na druhou stranu skromnější na množství informací, tedy může být vhodnější při rychlé a nepříliš podrobné analýze, ovšem nemusí obsahovat určité informace a indikátory škodlivého chování vzorku, pokud tyto aktivity nenechaly na systému žádné zřejmé změny (např. malware jež by skrytě bez uživatelova vědomí využíval procesorový čas k dolování krypto-měn by typicky nemusel zanechat žádné změny v systému, ovšem jeho chování je zřejmě nežádané). [19]

Jedna z věcí, na které se můžeme soustředit je například přístup k systémovým registrům. Program si do systémových registrů může ukládat různá data a konfiguraci, ovšem zapsáním vhodných hodnot do vhodných registrových záznamů může docílit zajištění spuštění (angl. persistence) vzorku, tedy zajistit, že daný vzorek malware se bude pravidelně spouštět například při startu systému, nebo třeba při určité akci jako přihlášení uživatele, spuštění Průzkumníka Windows atp. Potenciálních registrových záznamů jež by vedly k zajištění spuštění vzorku je mnoho, avšak zpravidla jsou snadno rozeznatelné od legitimně přistupovaných a vytvořených záznamů (popř. alespoň jejich názvy vzbuzují podezření a indikují potenciál nějaké nevhodné a možná škodlivé činnosti ze strany testovaného vzorku, když již přímo není zřejmé, že zajišťují spuštění kódu vzorku).

2.6 Sledování přístupu k souborům

Další významnou aktivitu vzorku malware jež je možno sledovat dynamickou analýzou je přístup vzorku k datovým souborům na discích dostupných operačnímu systému. Jak bylo popsáno již v kapitole 2.3.3, informace o tom jaké knihovny vzorek používá při svém běhu může poskytnout důležité informace o jeho potenciálních aktivitách. Proto skutečnost, že program načítá a používá nějaké další knihovny než jsou vypsány v importním adresáři v sekci `.idata` bude viditelná v podobě přístupu k těmto knihovnám. [19]

Toto je pouze jedna zájmová kategorie souborů jež jsou při forenzní analýze malware důležité. Podobně jako systémové registry i soubory mohou zajistit spuštění vzorku (angl. persistence), v případě že je vzorkem vytvořen soubor s kódem, jež zajistí žádanou škodlivou

činnost, v určité lokaci. Příkladem může být například soubor `autorun.inf` na přenosném médiu, jenž u starších verzí OS Windows umožnil automatické spuštění žádaného kódu. Podobně existují další možné soubory jež zajistí spuštění malware při nějaké aktivitě, jako například přihlášení uživatele (např. umístění vzorku do složky "Po spuštění"atp.).

Další z důvodů proč sledovat přístup malware k souborům je potřeba vědět jestli vzorek nemanipuluje se soubory, se kterými by manipulovat neměl. Ransomware například typicky bude přistupovat k souborům a přepisovat jejich obsah na jejich šifrovanou podobu. Principiálně tedy můžeme snadno rozpoznat tuto kategorii malware právě podle jejího přístupu k souborům. Podobně malware určený k exfiltraci dat bude typicky přistupovat k žádaným souborům a následně je ukládat na externí úložiště.

Konečně nás také zajímá přístup vzorku ke svým vlastním souborům. Především protože při analýze malware se snažíme zjistit co nejvíce podstatných informací o daném vzorku, je zřejmé že nás bude zajímat jaké informace si vzorek ukládá do svých vlastních souborů, ať už jsou to konfigurační soubory, soubory pro dočasnou paměť, či jiné datové soubory. Je zcela zřejmé, že sledování přístupu k souborům je z těchto všech důvodů kritické při forenzní analýze malware. [17]

2.7 Práce se semafory

Malware může považovat za nežádané, aby vícenásobně infikoval jeden systém. Například pokud by poslouchal na pevně stanoveném síťovém portu, pak by jednotlivé instance malware navzájem mezi sebou prováděly DoS na ostatní instance, v závislosti na tom kdo by právě získal přístup k danému portu. Aby tomuto autoři malware zabránili, zapojují do malware mechanismy k zabránění vícenásobné infekce systému. Typický příklad takového mechanismu jsou nabízející se semafory. Malware poté vytvoří a uzamkne semafor ve chvíli kdy infikuje daný systém a každé další spuštění daného malware zkontroluje dostupnost tohoto semaforu a v případě jeho existence se ukončí a neinfikuje systém. Alternativně může například malware zkoumat nejen přítomnost stejného vzorku, ale i přítomnost dalších, jiných, vzorků malware pomocí jejich vlastních semaforů a v případě, že žádaný vzorek malware na systému není aktivní, infikuje jej i tímto vzorkem. [15]

Z tohoto důvodu je pro vyšetřovatele zajímavé zjistit jestli malware pracuje nějakým způsobem se semafory a pokud ano, pak jakým způsobem a s jakými semafory. V případě že je název semaforu statický, je možné na jeho názvu založit detekci daného malware. Ovšem název semaforu nemusí být statický, pouze postačuje aby algoritmus tento název derivoval pomocí statického algoritmu z nějakého pevného atributu, jež je pevně stanovený pro každý systém avšak unikátní pro každé dva různé systémy. Jako příklad takového atributu, jež je využívám autory malware je produktové identifikační číslo OS Windows, jež se odvíjí od sériového čísla Windows. Toto číslo je zřejmě z licenčních důvodů pro každý systém jiný, pokud se ovšem nejedná o multilicenční kopii (což by poté pomohlo forenzním vyšetřovatelům, protože by se na všech takových systémech semafor jmenoval stejně a bylo by snadné zjistit zda jsou dané počítače infikované). [15]

2.8 Síťová aktivita vzorku

V případě že vzorek implementuje síťovou aktivitu, ať už s cílem stáhnutí kódu či komunikací s řídícím serverem (angl. Command and Control, typicky nazýváno zkráceně CC nebo C2), můžeme sledovat síťovou aktivitu vzorku a pokusit se z ní odvodit chování a účel

vzorku. Výhoda této metody je také v tom, že nemusíme mít žádné speciální nástroje (např. Wireshark) přímo na daném infikovaném systému, ale můžeme tyto nástroje umístit mimo systém, typicky na první síťový prvek, ke kterému je daný systém připojený. Sledování síťového provozu pak může probíhat zcela transparentně pro daný vzorek. [6]

Ovšem typicky při analýze malware nenecháváme systém na němž běží vzorek malware připojený k internetu, převážně protože předem nevíme jaká data by vzorek malware chtěl odesílat nebo přijímat, a s ohledem na skutečnost, že daný vzorek vykonává nežádanou škodlivou činnost, komunikovaná data by pravděpodobně také nebyla žádaná. Proto také moderní vzorky malware kontrolují dostupnost či nedostupnost jistých síťových zdrojů a v případě neočekávaného výsledku ukončí svou činnost, z obavy že jsou ve virtuálním prostředí ke zkoumání vyšetřovatelem malware, jak bylo již podrobně popsáno v kapitole 2.5. Z toho také vyplývá, že při automatické dynamické analýze se vzorek malware nemusí vůbec projevit, pokud nebyla nejprve důkladně prozkoumána a analyzována jeho síťová aktivita a ostatní metody dynamické analýzy jsou poté prakticky bezvýsledné. [6, 11]

Často navíc vzorky malware nekomunikují přímo se serverem pomocí pevně zakódované IP adresy, ale obsahují ve svém kódu doménové jméno, s nímž budou chtít komunikovat. Aby získali IP adresu ke které se mají připojit, musí tedy pomocí protokolu DNS [10] zjistit adresu příslušící danému doménovému jménu. Díky tomu vyšetřovatel dokáže odchytit zprávy DNS, zjistit jaké doménové jméno si vzorek vyžádal, zjistit o jakou doménu ve skutečnosti jde a navíc také odpovědět podvrženou DNS zprávou s podvrženou odpovědí - tedy adresou IP jež nemíří na skutečný server odpovídající této doméně, ale na předem připravený server, typicky sestavený vyšetřovatelem k podvrhnutí původního serveru, aby mohl vyšetřovatel určit jakým způsobem chce vzorek s daným serverem komunikovat (s jakou službou, na jakém portu, jaká data chce odesílat a přijímat atp.). [6, 11]

Kapitola 3

Existující nástroje k forenzní analýze malware

V této kapitole diskutujeme již existující nástroje určené k či užitečné při forenzní analýze malware. Jedná se o nástroje jež jsou každodenně používány forenzními vyšetřovateli k sestavení forenzních posudků vzorků malware. Zaměříme se především na otevřený volně dostupný software, ovšem zmíníme také významné volně dostupné (avšak uzavřené) nástroje.

3.1 Jednotlivé nástroje

V této podkapitole probereme jednotlivé nástroje, s jejichž použitím by forenzní vyšetřovatel dokázal plně sestavit kompletní forenzní posudek jakéhokoliv běžného vzorku malware. Tyto nástroje přímo implementují metody popsané v kapitole 2, nebo nepřímo umožňují popsané informace zjistit.

3.1.1 Strings

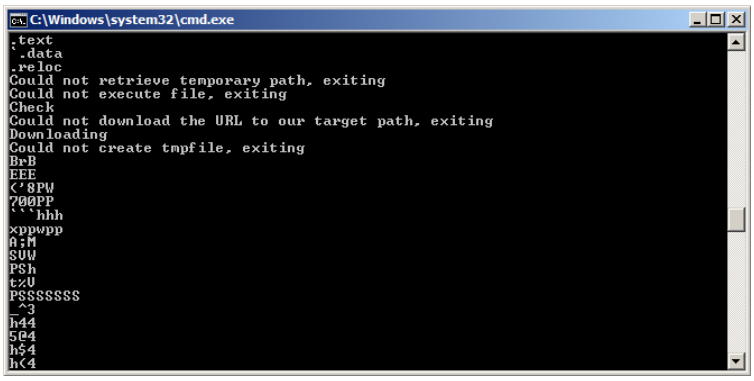
Nástroj Strings[1] od Sysinternals je základní nástroj implementující extrakci řetězců jak je popsána v kapitole 2.2. Tento nástroj je velmi základní a jeho cílem je především rozsáhlá kompatibilita. Podporuje kódování ASCII a Unicode a vypisuje všechny tisknutelné řetězce o délce větší než 3 (ve výchozím nastavení). Tento přístup nebude fungovat se šifrovanými řetězci a výstup navíc obsahuje značné množství nesmyslných řetězců, které nereprezentují žádný smysluplný řetězec. Příklad takového výstupu je k vidění na obrázku 3.1

3.1.2 PEiD

PEiD[2] je jeden z prvních nástrojů pro základní analýzu vzorku malware s primárním cílem zjistit zda je daný vzorek zašifrován či zkomprimován některým známým algoritmem či metodou. Oficiálně podpora a vývoj tohoto nástroje skončila, avšak stále je hojně využívaný při analýze malware, především proto, že jeho relevantnost neklesla a lze stále doplňovat databázi signatur, na jejichž základě jsou metody ve vzorcích identifikovány. Na základě tohoto nástroje vznikaly další nástroje s podobným cílem.

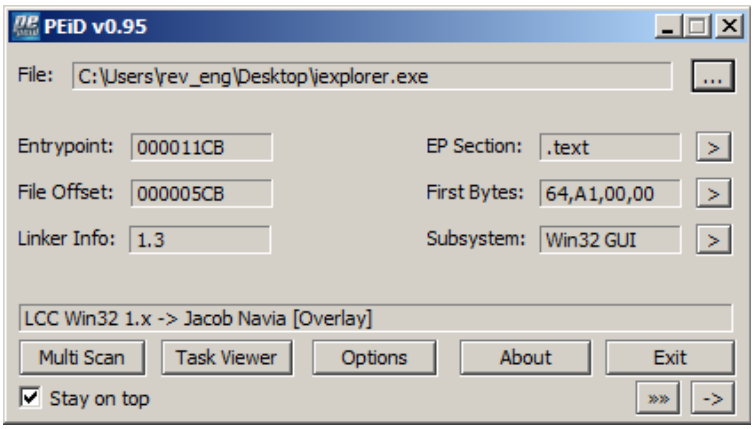
[1] Dostupný z `https://docs.microsoft.com/en-us/sysinternals/downloads/strings`
[2] Dostupný z `https://www.aldeid.com/wiki/PEiD`

Obrázek 3.1: Výsledky analýzy nástrojem Strings



Hlavní výhodou tohoto nástroje je právě identifikace konkrétní šifrovací metody, ihned tedy může vyšetřovatel na základě výsledků zjistit, jakým způsobem může data dešifrovat a nemusí ručně vyhledávat dešifrovací algoritmus v kódu a zjišťovat klíč apod.

Obrázek 3.2: Výsledky analýzy nástrojem PEiD



3.1.3 CFF Explorer

Nástroj CFF Explorer[3] umožňuje základní statickou analýzu v podobě extrakce znalostí ze záhlaví PE souboru, jak je popsáno v kapitole 2.3. Aplikace navíc obsahuje možnost základního překladu strojového kódu na asemblér a prezentaci získaného asembléru, avšak v praxi je vhodnější použít specializované nástroje, popsané v kapitole 3.1.8.

Hlavní výhodou tohoto nástroje je jeho kompletnost a exaktnost, výsledky totiž prezentuje skutečně kompletně včetně uvádění hodnot odsazení od počátku souboru, hexadecimální hodnoty a jejich význam u jednotlivých polí záhlaví PE, včetně informace jaké další hodnoty by dané pole mohlo obsahovat. Nevýhodou ovšem je, že tento program očekává že uživatel bude odborně zkušený a dokáže správně interpretovat vypsané hodnoty, neinformuje tedy uživatele o anomáliích, ani neprezentuje nijak speciálně výsledky vzhledem k jejich důležitosti či významnosti ve forenzní analýze, pouze kompletně prezentuje výsledky popořadě tak jak jsou zjišťovány.

[3]Dostupný z `http://www.ntcore.com/exsuite.php`

3.1.4 Process Explorer

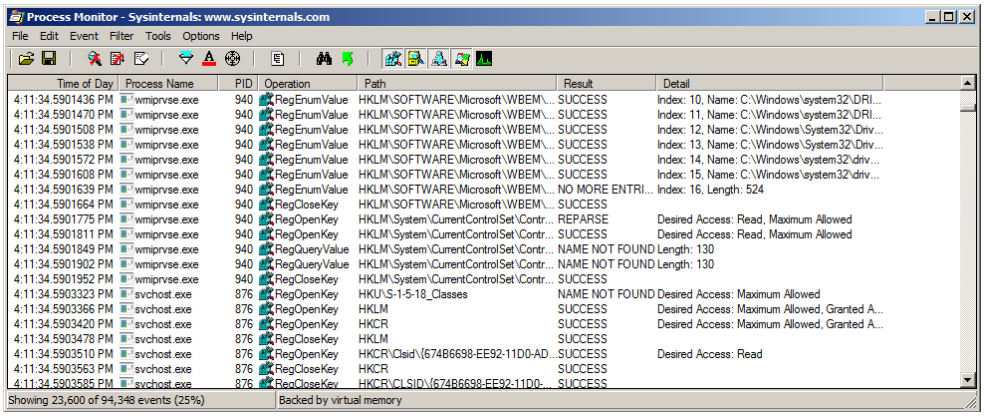
Tento obecný nástroj[4] od Sysinternals je opět velmi užitečný při dynamické analýze. Jedná se o pokročilou verzi správce úloh, jež je dodáván se všemi verzemi OS Windows. Navíc od běžně dodávaného správce úloh tento nástroj zobrazuje celou řadu dalších podrobných informací, například strom úloh (všechny podprocesy u každého procesu), podrobnosti o vláknech každého procesu, podrobnosti o síťových spojeních, existující semafory v systému, řetězce ve vzorku i v paměti alokované pro vzorek atd.

Ačkoliv tento nástroj není určený přímo pro analýzu malware, je při forenzní analýze velmi užitečný díky široké škále informací jež umožňuje zjistit. Nevýhodou je opět absence prezentace dat významných pro forenzní analýzu malware. Dále je nevýhodou, že je užitečný pouze při dynamické analýze, tedy při ručním použití během spuštění vzorku ve virtuálním prostředí, není tedy příliš vhodný pro automatickou analýzu, která je na místě pro přetížené týmy CSIRT.

3.1.5 Process Monitor

Process Monitor[5] je další nástroj od Sysinternals, který je opět velmi užitečný při dynamické forenzní analýze malware. Tento nástroj zaznamenává všechny přístupy procesů k systémovým zdrojům, tedy k souborům a registrům. Velmi detailně rozepisuje všechny pokusy o přístup všech programů. Hlavní nevýhodou je obrovské množství dat, ovšem za použití filtrů lze snížit toto množství na přijatelnou úroveň. Ve výsledku pak můžeme vidět nejen chování analyzovaného vzorku, ale i reakce ostatních běžících programů v systému. Lze tak snadno zjistit s jakými soubory a složkami proces pracuje, jaké procesy spouští a ukončuje (za použití jakých příkazů), atp.

Obrázek 3.3: Výsledky analýzy nástrojem Process Monitor



3.1.6 RegShot

Regshot[6] je volně dostupný otevřený software pro pořizování snímku systému. Umožňuje vytvořit snímky systému např. před a po spuštění vzorku malware a tyto dva snímky porovnat k získání seznamu změn mezi těmito dvěma snímky. Snímek není kompletním

[4] Dostupný z `https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer`
[5] Dostupný z `https://docs.microsoft.com/en-us/sysinternals/downloads/procmon`
[6] Dostupný z `https://sourceforge.net/projects/regshot/`

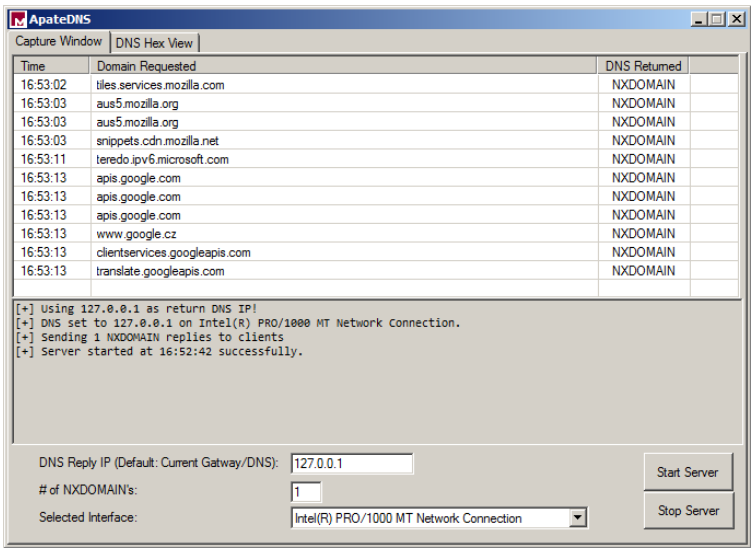
snímkem systému, ale soustředí se pouze na systémové registry a soubory na vybraných discích (či ve vybraných složkách).

Jak jsme popsali již v kapitole 2.5, množství informací takto získaných je podstatně menší než například u nástroje Process Monitor, což může být dobré (menší množství informací nezahltí vyšetřovatele zbytečnými informacemi), ale může to i uškodit – například pokud malware neprovádí trvalé změny v systému, kupříkladu odešle citlivá data přes síť, nepůjde toto ve výsledné zprávě nástroje RegShot vidět.

3.1.7 ApateDNS

ApateDNS[7] od Fireeye je užitečný program pro snadné monitorování a podvrhnutí dotazů protokolu DNS [10]. Jak již bylo zmíněno v kapitole 2.8, běžně bývá při dynamické analýze testovací prostředí odpojeno od internetu, proto ve chvíli kdy malware bude požadovat přístup k síťovému zdroji, mohl by na základě jeho nedostupnosti usoudit, že je spuštěn v bezpečném prostředí pro forenzní analýzu malware. Abychom se tomuto vyhnuli, můžeme pomocí tohoto nástroje zachytit všechny zprávy protokolu DNS a podvrhnout jejich odpověď, tak aby malware nabyl dojmu, že běží na reálném systému s běžným připojením do sítě internet. Nástroj navíc snadno umožňuje nejen číst zprávy DNS, ale také nastavit na jakou adresu mají být adresáti směřováni, na níž může běžet vyšetřovatelem předem připravený systém, který bude odpovídat na další požadavky zkoumaného malware.

Obrázek 3.4: Nástroj ApateDNS k podvrhnutí odpovědí DNS



3.1.8 Assemblérové debuggery

Ke statické analýze malware bezpochyby patří i nástroje jako IDAPro a OllyDbg, tedy nástroje jež umožňují forenznímu vyšetřovateli strojový kód binárního spustitelného souboru konvertovat do některého ze srozumitelných programovacích jazyků. Typicky se jedná o asemblér, avšak IDAPro umožňuje v placené verzi také konverzi do vysokoúrovňového pseudokódu podobnému jazyku C. Navíc tyto nástroje umožňují provádět debugging zkou-

[7] Dostupný z `https://www.fireeye.com/services/freeware/apatedns.html`

maných vzorků, konkrétně tedy spouštění vzorku krok po kroku (instrukce po instrukci), vytváření breakpointů, zkoumání obsahu paměti atp.

Tyto nástroje jsou pro forenzní vyšetřovatele skutečně kritické a jejich použitím se dochází k nejpodrobnějším a nejdůležitějším výsledkům forenzní analýzy, ovšem jejich použití vyžaduje vysokou odbornost a zkušenost v oboru forenzní analýzy a je velmi časově náročné. Navíc není možné je využít při automatické analýze. Navíc IDAPro v profesionální edici stojí kolem 40 tisíc Kč[8] za jednu licenci, přičemž tato profesionální edice přináší mnoho důležitých funkcí nedostupných v základní verzi IDA či OllyDbg. Z těchto důvodů týmy CSIRT nemají čas ani prostředky provádět analýzu pomocí těchto nástrojů a tyto nástroje jsou většinou používány spíše specializovanými laboratořemi a společnostmi na forenzní analýzu malware, zatímco týmy CSIRT se musí spolehnout na více automatizované a časově méně náročné nástroje.

3.2 Balíky nástrojů

V předchozí kapitole jsme se detailněji podívali na jednotlivé nástroje používané k forenzní analýze malware. Typicky forenzní vyšetřovatel musí využít několik těchto nástrojů a musí ručně analyzovat jejich výstup ve formě forenzního posudku. Použití těchto nástrojů požaduje určitou odbornost a zručnost a přestože výsledky bývají většinou přesnější, taková analýza zabere znatelně dlouhou dobu, což není akceptovatelné pro týmy CSIRT. Proto v této kapitole nahlédneme na existující balíky nástrojů, jež jsou sestaveny do kompletního řešení jež se automaticky snaží analyzovat daný vzorek malware mnoha různými metodami popsanými v kapitole 2.

3.2.1 VirusTotal

Pravděpodobně nejznámější online nástroj na analýzu malware je VirusTotal[9] od společnosti Google. Tento nástroj umožňuje nahrát nebezpečný soubor (či poskytnout `URL` adresu, vyhledat již dříve nahraný pomocí otisku hash atp.) a následně nad daným souborem provede plně automaticky forenzní analýzu, provede detekci za použití více než 60 antivirových programů. Tento nástroj poskytuje rychle a přehledně informace získané statickou i dynamickou analýzou v jednotném hlášení (viz obrázek 3.5) a implementuje skoro všechny metody popsány v kapitole 2.

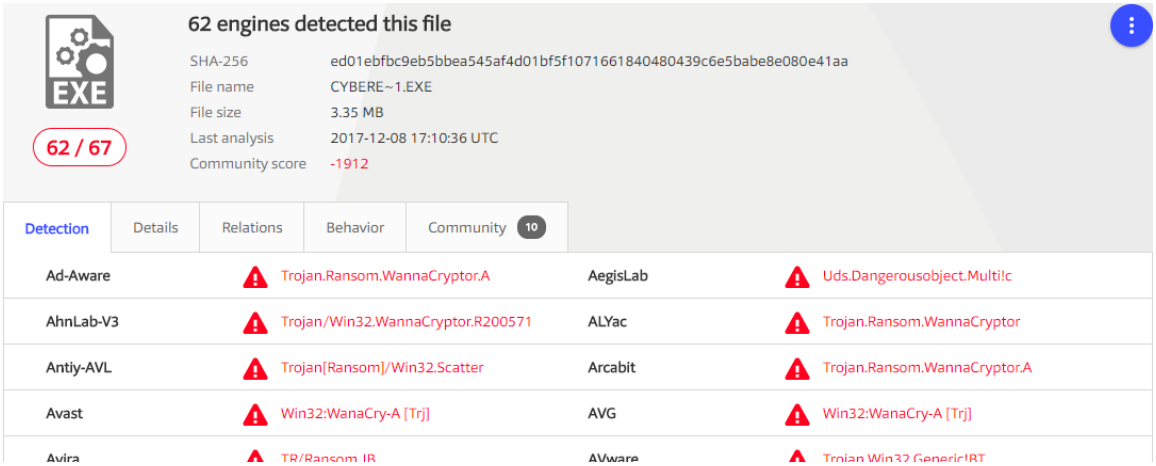
Nevýhodou tohoto nástroje je jeho uzavřený kód. Avšak hlavní nevýhodou, která zcela znemožňuje použití tohoto nástroje ve velkém množství případů je jeho otevřenost vůči komunitě. Specificky největší problém je například skutečnost, že všichni uživatelé mohou zobrazit hlášení o kdykoliv dříve zkoumaném vzorku. V případě že by autor malware upravil svůj malware tak aby měl unikátní otisk hash, tak že nikdy žádný vzorek s tímto otiskem pomocí nástroje VirusTotal zkoumán nebyl, pak ve chvíli kdy vyšetřovatel nahraje nalezený vzorek na VirusTotal, autor malware se ihned dozví, že jeho malware byl nalezen a je zkoumán forenzním vyšetřovatelem. Na tomto základě může například změnit strategii chování, vypnout vzorek atp.

Z tohoto důvodu, ačkoliv nástroj poskytuje opravdu velmi rozsáhlé funkce a spolehlivě integruje mnoho různých metod a nástrojů, není vhodné VirusTotal použít při vyšetřování forenzním týmem CSIRT v případech kdy byl detekován vzorek malware a není známo zda se jedná o starý běžně známý vzorek či o nový cílený útok (což není známo skoro v žádném

[8] Ceník IDA licencí je dostupný z `https://www.hex-rays.com/cgi-bin/quote.cgi`

[9] Dostupný z `https://www.virustotal.com`

Obrázek 3.5: Výsledky analýzy nástrojem VirusTotal



z incidentů). Existuje také mnoho dalších podobných nástrojů, avšak ze stejných důvodů je v této práci nebudeme dále diskutovat a zaměříme se především na otevřené nástroje, jež je možno využívat na svém systému, popř. virtuální sadě systémů.

3.2.2 Minibis

Nástroj Minibis[10], vyvíjen rakouským národním týmem CERT, je automatizovaný virtuální systém přizpůsobený pro forenzní vyšetřování. Tento nástroj je inspirován běžným postupem při základní dynamické analýze forenzního vyšetřovatele - tedy spuštění vzorku malware ve virtuálním prostředí a sledování jeho chování za pomocí speciálních nástrojů, jak je popsáno v kapitole 2.5. Nástroj Minibis toto automatizuje, tedy stačí tomuto nástroji poskytnout vzorek a ten se sám postará o sestavení nového virtuálního systému, nahrání vzorku, spuštění nástrojů a konečně automatický úklid a export hlášení v podobě CSV souborů a snímků obrazovky virtuálního systému během analýzy. Hlavní nevýhodou tohoto systému je jeho požadavek na běh na reálném HW s konkrétní instalací OS - Xubuntu. Další nevýhodou je že tento nástroj pouze provádí dynamickou analýzu, hlášení o výsledcích nejsou ve snadno prezentovatelném stavu (pouze obsahují surové výsledky nástrojů použitých k analýze a předpokládá se další analýza). Konečně tento nástroj sám o sobě neimplementuje žádné metody popsané v kapitole 2, ale pouze automatizuje použití nástrojů tyto techniky implementující. [23]

Ovšem tento nástroj je pro podstatu této práce důležitý, především protože je vyvíjen a používán národním týmem CERT, jejichž struktura a činnost je velmi podobná týmům CSIRT, jež jsou zaměřením této práce. Proto je tento nástroj také uváděn, i když přímo neimplementuje žádné metody popsané v kapitole 2.

3.2.3 Cuckoo Sandbox

Nástroj Cuckoo Sandbox[11] je velmi rozsáhlý volně dostupný otevřený nástroj pro analýzu malware. Tento modulární nástroj velmi připomíná funkcionalitou nástroje VirusTotal, ovšem je zcela otevřený a je možné jej spustit na vlastním stroji bez jakéhokoliv sdílení

[10] Dostupný z `https://cert.at/downloads/software/minibis_en.html`
[11] Dostupný z `https://www.cuckoosandbox.org/`

informací s třetími stranami. Tento nástroj je modulární konstrukce a obsahuje mnoho modulů, jenž implementují v různém rozsahu velkou část metod popsaných v kapitole 2. Hlavním cílem tohoto nástroje je dynamická analýza, která prováděna ve virtuálním prostředí za použití volně dostupné otevřené virtualizační platformy VirtualBox (popř. jsou v různé míře podporovány i další virtualizační platformy jako Xen či VMWare). Narozdíl od nástroje Minibis popsaného v předchozí kapitole však ve spuštěném virtuálním stroji nejsou spuštěny běžné forenzní nástroje (jako jsou popsány v kapitole 3.1), ale systém a zkoumaný vzorek je analyzován právě za pomocí forenzních modulů nástroje Cuckoo, jež jsou napsány v jazyku Python. Výstup v podobě technické zprávy je generován jako webová stránka za použití jazyka javascript a HTML.

Tento nástroj je nejblíže nástroji, jež budeme konstruovat v rámci této práce. Ovšem při implementaci našeho nástroje použijeme rozdílné postupy a přístupy k metodám získávání forenzních dat. Náš nástroj bude také jednodušší především z uživatelského hlediska, aby bylo ušetřeno co nejvíce času týmy CSIRT, jež by chtěli tento nástroj použít při analýze vzorků malware. Navíc výstup nástroje Cuckoo Sandbox v podobě webové stránky je velmi strohý a naopak podrobný technický výstup je extrémně detailní a navíc dostupný pouze ve strojovém formátu (konkrétně formátu `json`), takže není vhodný k běžné ruční analýze z hlediska časové a odborné náročnosti.

Kapitola 4

Analýza a specifikace požadavků

V této kapitole se v rychlosti seznámíme s týmem CSIRT a jeho rolí a motivací v organizaci a zaměříme se především na roli vyšetřovatele bezpečnostních incidentů.

4.1 Struktura a role CSIRT

Tým CSIRT[1] je organizace, nebo častěji organizační složka, jež zajišťuje několik kritických rolí v oblasti zabezpečení cílové oblasti. Typicky CSIRT vzniká na úrovni národní či organizační, ve chvíli kdy organizace uzná potřebu vlastního dedikovaného týmu pro řešení bezpečnostních otázek v organizační infrastruktuře či produktech. Velikost a schopnost týmu CSIRT se odvíjí od potřeb, jež má CSIRT zajistit. V případě drobných organizací mohou být úkoly spadající do oblasti činností prováděných týmem CSIRT úkolem jediného člověka, zatímco větší společnosti s více pobočkami mohou mít i desítky zaměstnanců.

Tým CSIRT má několik úkolů, jež typicky v organizaci zastává:

- Reakce na incidenty – Jak z názvu CSIRT vyplývá, hlavní role týmu CSIRT je reakce na bezpečnostní incidenty. To znamená přijetí hlášení o incidentu, analýza a klasifikace incidentu, vyřešení a uzavření incidentu.
- Proaktivní bezpečnost – Správa sítě spadající pod autoritu daného CSIRT z hlediska správy bezpečnosti, tedy typicky zařízení a správa bezpečnostní infrastruktury.
- Bezpečnostní monitorování – Průběžné monitorování sítě a zařízení spadajících pod autoritu CSIRT pro zajištění bezpečného fungování a plnění obchodních cílů organizace.
- Forenzní analýza – Forenzní analýza zařízení, systémů, malware a dalších artefaktů získaných při běhu CSIRT.

Dále také může CSIRT zajišťovat například zabezpečení produktu prodávaného danou organizací, tedy bezpečnostní správa včetně vyhledávání a opravování zranitelností, řešení bezpečnostních problémů atp. Dále může mít CSIRT v organizaci na starost osvětu bezpečnosti v informačních technologiích mezi zaměstnanci. [16]

Zaměřme se nyní na proces řešení incidentu a osobu, jež incident řeší a její práci, abychom v následující kapitole mohli pro tyto potřeby správně specifikovat návrh nástroje,

[1] CSIRT je zkratka pro Computer Security Incident Response Team

jež by pomohl při řešení incidentu zahrnujícího analýzu vzorku malware. Proces řešení bezpečnostního procesu začíná nahlášením podezřelé aktivity, ať se jedná o nahlášení osobou, nebo zachycení podezřelé aktivity sondou či senzorem. Následně je incident s veškerými iniciálními podklady předán (přidělen) vyšetřovateli. Vyšetřovatel získá všechny potřebné artefakty týkající se daného incidentu a začne s jejich analýzou, přičemž v průběhu analýzy může vzniknout potřeba získat další artefakty, které by pomohly incident vyřešit. Tento proces se opakuje dokud vyšetřovatel nemá všechny artefakty analyzované, tak aby byl schopný jednat na základě zjištěných informací a následně daný incident uzavřít. [16]

4.2 Specifikace požadavků

Jak bylo popsáno v předchozí kapitole, při řešení bezpečnostního incidentu v rámci CSIRT je třeba pracovat s různými artefakty. Jeden z takových artefaktů může být právě vzorek podezřelého kousku software, jež je třeba podrobit analýze s cílem identifikovat zda se jedná o malware a příp. jaký je jeho účel. Protože však takových vzorků může být při vyšetřování incidentu mnoho (a taktéž může být mnoho incidentů mnoha různých priorit), je třeba aby vyšetřovatel nevěnoval příliš mnoho času forenzní analýze malware. Je proto vhodné, aby nástroj byl jednoduchý nejen z hlediska uživatelské přívětivosti a použitelnosti, ale také aby nezahltil vyšetřovatele velkou spoustou informací. Navíc forenzní vyšetřovatel často nemusí být sběhlý ve forenzní analýze malware a proto pokročilá statická analýza a podobné pokročilé expertní techniky nemohou být od vyšetřovatele očekávány. Proto na nástroj pro podporu rozhodování při řešení bezpečnostního incidentu v CSIRT, jež tvoříme v této práci, musí splňovat následující předpoklady:

- Jednoduchá použitelnost – Nástroj musí mít uživatelsky přívětivé grafické prostředí jež je jednoduché a přehledné. Práce s uživatelským prostředím musí být intuitivní a prostředí musí být snadno použitelné. Nesmí být nutné učit se jakékoliv příkazy či učit se vnitřní pochody nástroje. Stručně řečeno, osoba mající základní znalost v oblasti vyšetřování incidentů musí být schopna nástroj požít bez studování jakýchkoliv postupů či návodů. Důvodem k tomuto požadavku je především absence potřeby tréningu personálu, což podpoří využitelnost nástroje v prostředích jako je právě CSIRT, kde není prostředků nazbyt. V kapitole 3 byly popsány nástroje, jež jsou velmi mocné, avšak týmy CSIRT je nepoužívají třeba právě kvůli jejich neintuitivnosti a obtížné použitelnosti, protože vyšetřovatelé by nejdříve potřebovali rozsáhlý tréning a velmi by jejich použití zpomalovalo v situacích, kdy je třeba jen rychlá základní analýza.

- Jednoduchý výstup – Výstup nástroje musí být přehledný a jednoduchý. Nástroj nesmí vyšetřovatele zahrnout spoustou odborných informací, ale informace musí být podány přehledně a srozumitelně. Je vhodné použít grafický výstup k částem analýzy, kde toto dává smysl a podpoří to srozumitelnost výsledků. Důvodem k tomuto požadavku je podobně jako u předchozího požadavku zvýšení přívětivosti k vyšetřovatelům a použitelnosti nástroje bez jakéhokoliv předchozího tréningu. Vyšetřovatel se základními znalostmi v oblasti analýzy malware by měl být schopný interpretovat prezentované výsledky a neměl by potřebovat žádné další nástroje aby získal, doloval či agregoval prezentovaný výstup s cílem získat užitečné znalosti sám.

- Základní implementace – Nástroj musí splňovat některé minimální požadavky na implementaci, tedy implementovat minimálně takové metody, které umožní reprezentaci následujících informací:

  - Základní informace o vzorku – Nástroj musí zobrazovat základní informace o vzorku, zjistitelné ze záhlaví Portable Executable.
  - Pokročilé informace ze záhlaví PE – Nástroj musí dále zpracovávat zajímavé podrobnější informace ze záhlaví Portable Executable netriviálním způsobem, především např. statisticky významná data prezentovat vhodným způsobem.
  - Informace o binárních řetězcích – Nástroj musí být schopen nalézt a vhodně prezentovat řetězce nacházející se v kódu zkoumaného vzorku.
  - Základní informace ze statické analýzy kódu – Nástroj musí být schopen provést na základní úrovni automatickou analýzu kódu vzorku a vhodným způsobem informovat o informacích získaných ze statické analýzy kódu vzorku malware.

- Rozšiřitelná implementace – Nástroj musí být možné jednoduše rozšířit o další vlastnosti a metody. Implementace musí umožňovat modulárně přidávat funkcionalitu nástroje bez nutnosti jakékoliv změny do existujícího kódu aplikace. Důvod k tomuto požadavku je zřejmý – obor forenzní analýzy malware je velmi dynamický a rok od roku autoři malware přichází s novými postupy jak maskovat malware, s novými zranitelnostmi a metodologiemi. Zabudování nástroje do metodik řešení incidentu týmem CSIRT je nelehký a nákladný úkol přestože se jej tento nástroj snaží co nejvíce zjednodušit. Ovšem aby byl tento nástroj použitelný a schopný plnit svůj úkol i za několik let, musí být možno jej v průběhu snadno modifikovat a vylepšovat, aby stále odpovídal aktuálním moderním metodologiím používaným ve forenzní analýze malware.

- Bezpečnost – Nástroj musí být bezpečný, není přípustné aby zkoumaný malware mohl jakkoliv poškodit systémy na nichž běží, či změnit nebo přistoupit k informacím k nimž nemá mít přístup. Důvod k tomuto požadavku je taktéž zřejmý – CSIRT na svých systémech z podstaty svého cíle uchovává mnoho citlivých a kritických dat, proto je naprosto neakceptovatelné, aby tento nástroj mohl být hrozbou pro tyto informace a data.

- Centrálně spravované – Nástroj by mělo být možné spravovat centrálně, ideálně v takové míře, aby instalace a konfigurace nástroje ze strany vyšetřovatele byla minimální či žádná. Důvodem k tomuto požadavku je podobně jako u předchozích bodů maximální usnadnění integrace nástroje do procesu řešení incidentů. V ideálním případě by byl nástroj poskytován jako interní služba, nainstalovaná a spravovaná správcem zázemí a vyšetřovatel by ji pouze začal používat. Tímto by nadále klesla nutnost tréningu vyšetřovatelů a tím pádem i náklady na integraci tohoto nástroje do procesů týmu CSIRT.

Kapitola 5

Návrh

V této kapitole specifikujeme technický návrh nástroje na základě analýzy a specifikace požadavků z kapitoly 4. Nejdříve se zaměříme na návrh nástroje tak, aby splňoval požadavky nastíněné v kapitole 4 a následně navrhneme ukázkový případ užití nástroje v průběhu řešení bezpečnostního incidentu týmem CSIRT.

5.1 Struktura a metodologie

Abychom co nejvíce vyhověli striktním požadavkům, nástroj navrhneme jako službu běžící na serveru. V dnešní době se stále více a více rozvíjí užití virtualizace a virtuálních serverů, především díky jednodušší správě, nižším nákladům a dalším specifickým výhodám virtualizačních technologií. Proto nástroj navrhneme jako službu jež může běžet právě na virtuálním serveru. Důvodem k tomuto požadavku je nejen užití moderních technologií, ale také zvýšení bezpečnosti - pokud by z jakéhokoliv důvodu analyzovaný malware dokázal ovlivnit systém na němž je zkoumán, nedojde k žádným významným škodám, protože by nástroj běžel na speciálním připraveném dedikovaném virtuálním stroji.

Protože požadavkem na nástroj je snadné použití nástroje a ideálně žádná instalace či konfigurace nástroje ze strany vyšetřovatele, navrhneme tento nástroj jako webovou službu. Toto je zdrojem několika hlavních přínosů:

- Absence klienta – protože jsme nástroj navrhli jako serverovou aplikaci jež poskytuje funkcionalitu nástroje jako službu, je třeba k této službě zajistit přístup pro vyšetřovatele. Díky navrhnutí nástroje jako webovou službu, nebude třeba aby vyšetřovatel na svůj počítač instaloval klienta a jednoduše bude moci k nástroji přistoupit ihned za použití standardního webového prohlížeče a bez potřeby jakéhokoliv dalšího kroku.
- Uživatelská přívětivost – Webová služba poskytuje možnost jednoduchého, přehledného a intuitivního uživatelského prostředí. U již existujících nástrojů jako je VirusTotal popsaný v kapitole 3.2.1 vidíme že tento přístup je možný a efektivní.
- Multiplatformní užití – Vyšetřovatelé mohou využít jakýkoliv oblíbený operační systém a není třeba vyvíjet klienta pro různé platformy a operační systémy.

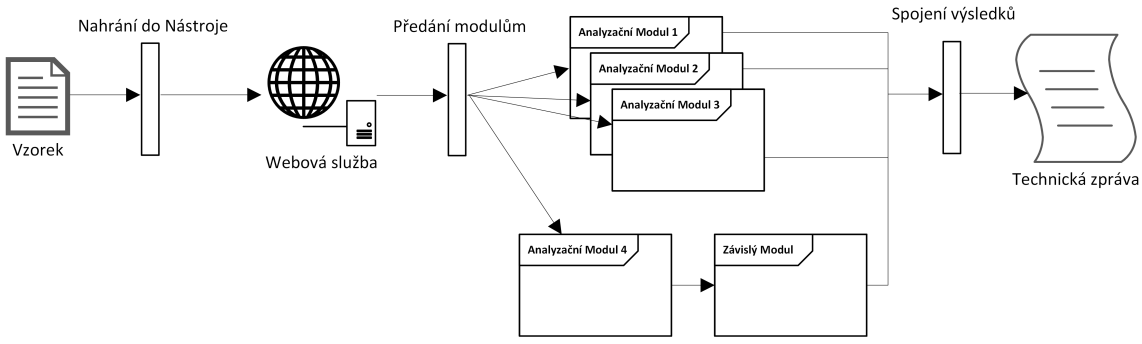
Tuto webovou službu by bylo vhodné poskytnout pouze autorizovaným osobám. Nástroj však záměrně navrhneme tak, aby neimplementoval žádnou autentizaci či autorizaci. Důvod je snadný – každý CSIRT je zpravidla součástí zaběhlé infrastruktury, jež již obvykle nabízí

autentizační a autorizační služby napříč celou organizací, jako například služby federovaného přihlášení založené na LDAP či Shibboleth. Implementace vlastní autentizační služby by tedy bylo spíše překážkou pro mnoho týmů CSIRT. Navíc v návrhu nástroje neuvažujeme že by nástroj ukládal jakákoliv data, proto webová aplikace nebude obsahovat žádnou databázi, kde by standardně byly přávě přihlašovací údaje uloženy. Nástroj bude spuštěn pro každý vzorek samostatně a mezi jednotlivými analýzami vzorků nebudou ukládány a sdíleny žádné informace, není tedy třeba mít žádnou databázi k ukládání jakýchkoliv dat.

Na implementaci zvolíme vysokoúrovňový jazyk Python, protože se jedná o jazyk široce využívaný, moderní, vysokoúrovňový a především vhodný k implementaci tohoto nástroje. K implementaci využijeme ve vhodné míře knihovny dostupné pro tento jazyk, abychom nemuseli vytvářet znovu věci jež jsou již hotové a veřejně dostupné v kvalitní, otestované implementaci. Užití existujících knihoven nám navíc umožní kvalitně nástroj implementovat s co největší funkcionalitou a umožní soustředit se na věci kritické pro forenzní analýzu a implementaci potřebných metod. Proto je nasnadě využít knihoven pro tvorbu webového prostředí (výstupu HTML) a podobně.

Abychom dosáhli požadované rozšiřitelnosti nástroje, implementujeme nástroj tak, aby jeho funkcionalita byla modulární. Tedy veškerá existující funkcionalita (v našem případě jednotlivé metody a postupy forenzní analýzy) byla rozčleněna do jednotlivých separátních modulů. Každý modul bude reprezentovaný vlastním souborem. Moduly budou navíc společně v jediné složce k tomuto určené. Další moduly v budoucnu vytvořené budou muset pouze splňovat předepsanou strukturu a v jednotlivých souborech budou uloženy v této složce, což umožní jejich načtení a užití při používání nástroje. Díky tomuto modulárnímu přístupu splníme požadavek na snadnou rozšiřitelnost funkcionality nástroje.

Obrázek 5.1: Diagram datového toku v nástroji

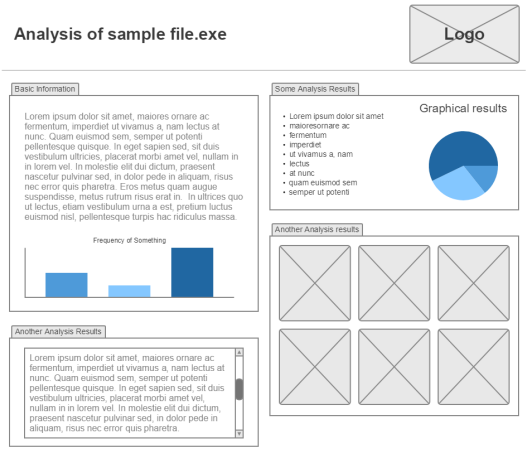


Protože hlavní podstata nástroje bude analýza malware, kde jednotlivé kroky analýzy jsou implementované v jednotlivých modulech jak bylo popsáno výše, samotná aplikace využívající tyto moduly bude poměrně jednoduchá a implementovatelná v několika málo souborech. Moduly budou naopak nosnou implementací. Jednotlivé moduly budou vycházet ze šablony rodiče `AnalysisModule`, jež bude abstraktní třída popisující API jednotlivých metod nutně implementovaných potomky této třídy. Mezi tyto metody bude patřit především metoda `Analyze` jež bude hlavní kostrou každého modulu – tato metoda totiž bude volána hlavním programem a v parametru bude předán odkaz na soubor určený k analýze.

Obrázek 5.2: Náčrt vzhledu webového rozhraní zprostředkujícího práci s nástrojem

(a) Náčrt úvodní strany k nahrání vzorku

(b) Náčrt stránky s výsledky analýzy



Výstupem této funkce budou 2 stěžejní věci:

- Výstup analýzy modulu – Jendotlivé moduly budou prezentovat své výsledky ve formě HTML výstupu, jež bude poté společně s ostatními moduly spojeny pro získání finální stránky s celkovými výsledky analýzy.
- Informace pro naplnění Knowledgebase – Všechny výsledky analýzy (a potenciálně i další neprezentované) budou v surové, strojově čitelné formě dostupné ostatním modulům. Tyto výsledky budou dostupné v průběhu této jediné analýzy v úložišti nazývaném Knowledgebase, odkud mohou být použity dalšími moduly k další analýze. Díky tomuto budou moci být jednotlivé moduly závislé a výsledky analýzy jednoho modulu budou moci být použity i dalším modulem ať už ke zpřesnění, agregaci, či třeba uložení výsledků do souboru.

Díky systému Knowledgebase popsanému výše, bude zajištěna ještě snadnější rozšiřitelnost funkcionality v budoucnosti, především v situacích kdy existující modul již nesplňuje všechny potřebné požadavky, bude možné z jeho analýzy vycházet a jen vytvořit modul který dále s těmito daty bude dále pracovat bez potřeby opět implementovat jeho funkcionalitu či upravovat jeho kód. Taktéž lze vytvořit např. moduly jež nebudou mít žádný výstup a jejich jedinou rolí bude třeba zjištění určitých informací a nahrání do Knowledgebase, odkud tyto budou použity jinými moduly. Díky tomu lze dosáhnout ještě větší modulárnosti a v případě potřeby rozčlenit rozsáhlé moduly do menších, lépe spravovatelných celků.

Vzhledem k tomu však může vznikat závislost na jednotlivých modulech - tedy pro správnou funkčnost modulu A je třeba mít naplněnou Knowledgebase informacemi jež získá modul B, je proto třeba zajistit, aby modul B byl spuštěn před modulem A. Navíc kvůli tomuto systému může dojít k uváznutí (angl. deadlock), jestliže by modul A jako svou prerekvizitu měl data získaná modulem B a ten naopak potřeboval data modulu A. Je třeba na toto pamatovat a navrhnout nástroj takovým způsobem, aby v takovémuto uváznutí nemohlo dojít (např. detekcí uváznutí a nespuštění daných modulů namísto uváznutí v nekonečném cyklu).

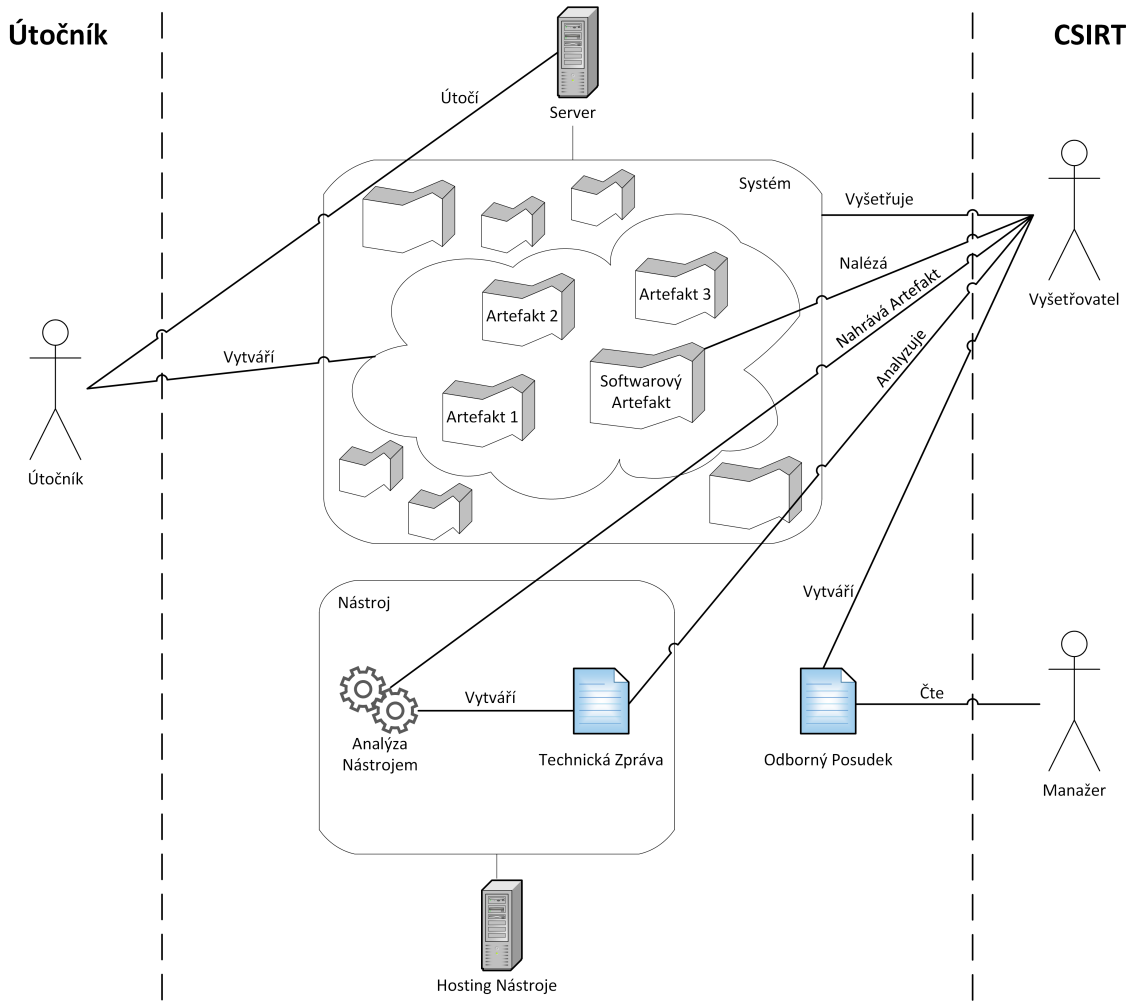
Úvodní stránka (náčrt k vidění na obrázku 5.2a) bude sloužit k nahrání vzorku do systému, což půjde učinit buďto přetáhnutím vzorku do určeného prostoru (angl. ztv. Drag

and Drop), popř. vyhledat a nahrát soubor ve svém počítači pomocí standardního dialogu pro nahrání souboru. Po stisknutí tlačítka `Upload and Analyze` dojde k nahrání souboru na server kde služba běží a poté na tomto souboru bude postupně spuštěna analýza pomocí jednotlivých modulů. Každý modul může poskytnout výstup ve formátu HTML, jež bude ve vlastním bloku přidán do celkové stránky s výsledky, podobně jako je načrtnuto na obrázku 5.2b. Celkový diagram toku dat je k vidění na obrázku 5.1.

5.2 Případ užití

Na obrázku 5.3 je k vidění diagram typického plánovaného případu užití nástroje jehož návrh je popisován v této kapitole. Tato typická situace začíná u útočníka, v diagramu vyobrazen vlevo, jež útočí na server pod bezpečnostní správou týmu CSIRT, jež je v diagramu vyobrazen na pravé straně. Při tomto bezpečnostním incidentu útočník svou činností dává vzniku mnoha artefaktům na systému, na něž následně vyšetřovatel při vyšetřování narazí. Mezi těmito artefakty může být i softwarový artefakt, jako například podezřelá aplikace umístěná na systému útočníkem.

Obrázek 5.3: Diagram případu užití



Aby vyšetřovatel dokázal v rámci vyšetřování rychle rozhodnout o roli a potenciálu daného softwarového artefaktu, nahraje tento nalezený artefakt do nástroje, jež provede automatickou analýzu všemi implementovanými metodami a v podobě odborné technické zprávy informuje vyšetřovatele o technických parametrech daného vzorku. Vyšetřovatel poté následně na základě této technické zprávy určí důležitost a roli daného vzorku v rámci vyšetřovaného incidentu a na základě těchto zjištění společně s ostatními artefakty výsledky vypracuje finální odborný posudku určený pro manažery týmu CSIRT, jež na jeho základě dále rozhodnou o dalším jednání v rámci vyšetřování tohoto incidentu.

Kapitola 6

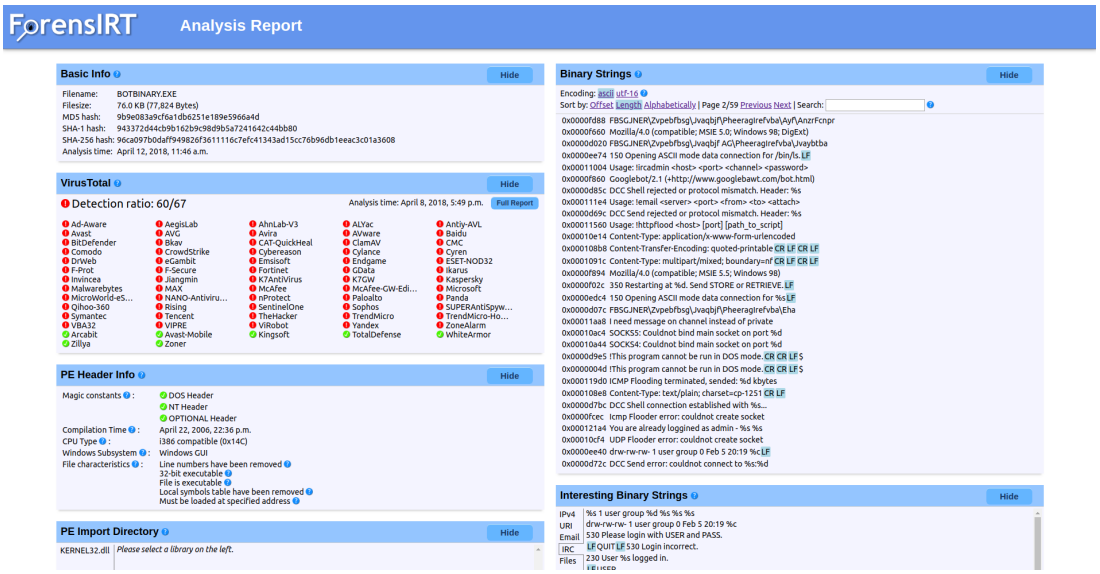
Implementace

V této kapitole nahlédneme na implementaci nástroje implementovaného v rámci této diplomové práce. Nejdříve se v kapitole 6.1 zaměříme na design a vytvoření značky nástroje. V následujících kapitolách se již zaměříme na technickou implementaci, použité technologie a nástroje. Závěrem této kapitoly prozkoumáme implementační detaily nástroje, metodologii, návrh a implementaci modulárního systému analyzačních modulů jako hlavní kostry nástroje.

6.1 Design

Design nástroje vychází z návrhu designu v kapitole 5. Tedy design je především minimalistický a funkční. Úvodní stránka nabízí pouze to nejnutnější, tedy prostor pro nahrání souboru k analýze tímto nástrojem. Stránka obsahující technickou zprávu dle návrhu rozděluje výstup do jednotlivých bloků rozřazených do dvou sloupců, kde každý blok slouží jako logické ucelení výstupu jedné analyzační metody.

Obrázek 6.1: Snímek výstupní technické zprávy nástroje ForensIRT



Design nástroje je založen na pevné šířce stránky (nikoliv na dynamické - závislé na rozlišení prohlížeče) o šířce 1720 pixelů, tedy správně a plně se zobrazuje na displejích

o rozlišení Full HD `1920x1080` a více. Na menších displejích bude zobrazena pouze část stránky v horizontální rovině (zbytek bude dosažitelný pomocí horizontálního posuvníku v prohlížeči). Tato nadstandardně velká pevná šířka stránky byla zvolena především na základě následujících argumentů:

- Nadstandardní technická vybavenost cílové skupiny – Vzhledem k cílové skupině tohoto nástroje, tedy bezpečnostním týmu operujícím v informačních technologiích a orientujícím se v současných trendech, lze předpokládat minimálně standardní technickou vybavenost, tedy displeje a zobrazovací zařízení minimálně s rozlišením Full HD (`1920x1080`) a vyšším. Toto je především založeno na skutečnosti že takový bezpečnostní tým má za úkol bojovat s aktuálními bezpečnostními hrozbami vycházejícími z aktuálních trendů a technologií, tedy lze také očekávat, že technologické vybavení takového bezpečnostního týmu bude odpovídat jeho potřebám vycházejících z této skutečnosti, tedy bude relativně nadstandardní.
- Větší prostor pro zobrazení dat – Zvýšením šířky stránky na nadstandardních 1720 pixelů se zvětší horizontální prostor k zobrazení dat. Tedy je možné zobrazit širší datové tabulky a podobné horizontálně zaměřené objekty. Důležitost tohoto aspektu je umocněna volbou dvousloupcového výstupu nástroje, jež samo o sobě vede ke značnému zúženi horizontálního prostoru k vykreslení dat výstupu jednoho analyzačního modulu.

Design nástroje je založen na odstínech modré barvy. Pozadí obsahu analyzačních modulů je pro podporu čitelnosti textu bílé a ostatní prvky na stránce jako záhlaví stránky, tlačítka apod. jsou v různých, podobných odstínech modré barvy. Tato volba vede k pocitu sjednocení vzhledu stránky.

Důležitým aspektem je také název nástroje. Na název nástroje byly kladeny následující požadavky:

- Snadno čitelný – Název musí být snadno čitelný v anglickém jazyce.
- Snadno zapamatovatelný – Název musí být snadno zapamatovatelný.
- Unikátní – Název nesmí být již používán jiným nástrojem, uskupením, či obecně jiným objektem či entitou.
- Tématický – Název musí obsahovat či vycházet ze slova tématicky příbuzného (např. malware, forensic sciences, incident response team, security...).

Na základě těchto požadavků byl vybrán název ForensIRT kombinující slova *Forensic* a *IRT*, tedy angl. zkratku pro *Incident Response Team* (přeložitelné do češtiny jako Skupina reakce na incidenty). Pro potřeby nástroje bylo dále navrhnuto logo nástroje (obr. 6.2).

Obrázek 6.2: Logo vyvinutého nástroje ForensIRT

Jako jazyk nástroje byla zvolena angličtina. Důvod k této volbě byly především následující argumenty:

- Komunikace bezpečnostních týmů – Bezpečnostní týmy existují často také na národní či nadnárodní úrovni. Jedním z úkolů bezpečnostního týmu také často bývá komunikace a spolupráce s ostatními (často zahraničními) bezpečnostními týmy, popř. také vydávání veřejných hlášení o zkoumaných vzorcích malware (technické zprávy a forenzní rozbory). Proto je vhodné mít výstup nástroje již v nějakém mezinárodně užívaném jazyce.

- Vysoký výskyt nepřekládaných odborných termínů – Forenzní analýza malware obsahuje velký počet angl. odborných termínů, pro které v češtině neexistuje překlad, popř. se častěji užívá angl. originální termín. Z tohoto důvodu, a také z důvodu ucelení technické zprávy, je vhodné celý text technické zprávy uchovávat v angličtině.

- Mezinárodní využití – V české republice existuje jen pár desítek bezpečnostních týmů. Aby byl nástroj otevřen širšímu publiku, je vhodné aby byl využitelný i v nějakém mezinárodním jazyce, např. v angličtině.

6.2 Užité technologie

Nástroj byl dle návrhu popsaného v kapitole 5 implementován v jazyce Python, konkrétně verze 3. Při vývoji bylo použito několik různých knihoven a frameworků dostupných pro tento jazyk. Tato kapitola popisuje tyto knihovny a nástroje, způsob a důvody jejich použití a specifické vlastnosti a konfiguraci těchto nástrojů a knihoven.

6.2.1 Framework Django

Django[1] je framework pro vývoj webových aplikací v jazyce Python. Tento framework umožňuje rychle vyvíjet webové aplikace při zachování čistého a strukturalizovaného kódu, společně s využíváním moderních návrhových vzorů a metodologií. Dalším důležitým přínosem je poskytnutí webového serveru frameworkem Django, což umožňuje věnovat se při implementaci nástroje pouze vývoji samotné aplikace a není třeba zbytečně věnovat čas tvorbě vlastního webového serveru. Django je navíc populární, otevřený a komunitou dobře podporovaný a stále vyvíjený nástroj. Především z těchto důvodů byl zvolen jako základ pro vývoj nástroje, v závislosti na požadavku aby výstup aplikace v podobě technické zprávy byl ve formátu HTML a nástroj byl dostupný jako webová služba, jak bylo popsáno v kapitole 5.

Aplikace implementované za použití frameworku Django jsou strukturalizované do návrhového vzoru inspirovaného vzorem `Model View Controller` (často zkracováno `MVC`). Ovšem Django implementuje tento návrhový vzor v mírně pozměněném stavu - `Model Template View` (`MTV`). `Model` je zdrojem dat webové aplikace, ve většině případů tedy například databázový systém. `Template` se stará o vykreslení zpracovaných dat do finální podoby - tedy do webové stránky ve formátu HTML. Konečně `View` zajišťuje propojení předchozích dvou součástí do jediného celku.

Ovšem protože nástroj implementovaný v rámci této diplomové práce je navržen jako systém bez využití databázového systému, musela být při implementaci vynechána právě část vzoru `Model` tak jak je implementován a poskytován frameworkem Django. Ve výchozím stavu totiž Django automaticky vytváří a spravuje databázi na základě definovaných

[1] Dostupný z `https://www.djangoproject.com/`

tříd a jejich atributů. Tedy všechny třídy jež dědí třídu `django.db.models.Model` jsou automaticky frameworkem využity k vytvoření příslušných tabulek a správě obsahu databáze obecně. Tato část frameworku Django byla tedy vynechána a nahrazena vlastní implementací vzoru `Model`, jež nedědí dříve zmíněnou třídu `django.db.models.Model` a tudíž se na ni nevztahují s tím spojené automatické úkony práce s databází. I přes tuto zásadní změnu nově implementovaný model stále zachovává původní logiku vycházející z návrhového vzoru `Model Template View`, tedy oddělení logické části kódu jež zajišťuje zdroj dat jež budou v aplikaci zobrazeny - v případě tohoto nástroje je zdrojem dat získaných analýzou vzorku malware. Jinými slovy právě tento vlastní speciální `Model` zajišťuje analýzu vzorku a následně předání získaných dat `View` která jsou následně vykreslena pomocí `Template`.

6.2.2 Knihovna pefile

V rámci implementace nástroje ForensIRT byla také využita knihovna pefile[2]. Tato knihovna si klade za cíl analyzovat soubory ve formátu Portable Executable a výstup této analýzy, především údaje extrahovatelné ze záhlaví formátu Portable Executable, poskytnout ve strukturalizovaném stylu jako instanci třídy `PE` jejíž atributy jsou právě hodnoty získané ze záhlaví PE. Tato knihovna navíc podporuje více typů souborů PE než pouze spustitelné soubory platformy Windows 32bit (tedy nejčastější spustitelný soubor na této platformě), jako například Windows 64bit či také binární EFI soubory a další. Dále tato knihovna podporuje analýzu binárních souborů PE pro různé architektury a je nezávislá na endianitě analyzovaných souborů.

Díky využití této knihovny bylo možné při implementaci projektu věnovat maximum času vývoji podstatnějších částí a implementovat celou řadu dalších analyzačních metod. Knihovna úspěšně vyřešila problém nalezení atributu na správném odsazení v binárním souboru (nebylo tedy třeba dle technické dokumentace formátu PE dohledávat potřebné atributy a jejich umístění v záhlaví PE), které navíc mohlo být dynamické – závislé na přechozím obsahu, například délce programu v hlavičce DOS. Dále se knihovna postarala o správné načtení hodnoty daného atributu v závislosti na endianitě a obecně cílové architektuře daného souboru.

Hodnoty získané knihovnou pefile jsou v nástroji použity v řadě analyzačních modulů, především v modulech týkající se statické analýzy.

6.2.3 Nástroj YARA

Nástroj YARA[3] vyvíjený společností Google je nástroj k vyhledávání vzorů (angl. pattern matching) v binárních souborech. Tento nástroj byl vytvořen a vyvíjen speciálně pro potřeby forenzní analýzy malware, je proto k těmto účelům vhodnější než ostatní knihovny určené k vyhledávání vzorů v binárních souborech. Tento nástroj je založen na databázi pravidel, jež obsahují různé definice (jednoduché, složené, podmíněné a další) textových či binárních dat jež mají být v analyzovaných vzorcích vyhledány. Tato pravidla zpravidla obsahují signatury známých vzorků malware, ale i signatury dílčích částí, jako například kryptografické algoritmy určitých skupin (útočníci jako např. Advanced Persistent Threat (APT) často opakovaně využívají úseky kódů jako např. vlastní implementace kryptografických algoritmů, což umožňuje detekci budoucích vzorků malware od těchto skupin), ale i signatury obecně užívaných úseků kódu (např. úsek kódu umožňující obejití antivirového

[2] Dostupná z `https://github.com/erocarrera/pefile`
[3] Dostupný z `https://github.com/VirusTotal/yara`

programu apod.) s cílem umožnit poodhalení schopnosti vzorku bez nutnosti nahlédnout do zdrojového kódu aplikace (což bývá zpravidla náročné a zdlouhavé).

V rámci implementace nástroje byla dále použita veřejně dostupná databáze pravidel zvaná YARA Rules[4], jež obsahuje stovky signatur rozdělených do logických kategorií jako signatury k identifikaci exploit kitů, kryptografických algoritmů, kompresních algoritmů (tzv. packerů), anti-vm a anti-debugging technik a dalších. Společně s touto databází byl nástroj využit v modulu v rámci statické analýzy vzorku.

6.2.4 Cuckoo Sandbox

Nástroj Cuckoo Sandbox byl popsán již dříve v kapitole 3.2.3. Tento nástroj je přední otevřený nástroj pro dynamickou analýzu malware, velmi dobře implementuje správu virtualizované infrastruktury a analýzu vzorků software na této infrastruktuře. Možnosti nástroje Cuckoo Sandbox jsou velmi rozsáhlé a nástroj dokáže efektivně spravovat infrastrukturu o jednom virtuálním stroji na otevřené virtualizační platformě VirtualBox až po obrovskou decentralizovanou infrastrukturu různých virtuálních strojů a sítí na komerčních virtualizačních platformách jako např. VMWare. Ovšem jeho použitelnost samostatně je spíše omezená, především proto, že jeho výstup v podobě webové stránky je velmi stručný a obsahuje jen zlomek informací vzhledem k tomu kolik informací svou automatickou dynamickou analýzou získá. Navíc statickou analýzu provádí spíše na základní úrovni.

Protože je Cuckoo Sandbox velmi dobře implementovaný ke správě virtualizační infrastruktury a k dynamické analýze malware, pak s ohledem na skutečnost, že dynamická analýza malware (ze své podstaty spouštění nebezpečného kódu) je velmi nebezpečná, bylo rozhodnuto, že bude Cuckoo Sandbox využit k dynamické analýze malware, především že se jedná o dlouhodobě vyvíjený projekt s otevřeným kódem, který je navíc dobře známý a již několik let má velmi dobrou komunitní podporu. Bylo tedy rozhodnuto, že je lepší využít existující, zřejmě kvalitní komunitní nástroj, než vytvářet v rámci této práce nový, znatelně omezenější a potenciálně nebezpečnější nástroj se stejným cílem, jako poskytuje Cuckoo Sandbox.

Cuckoo Sandbox nabízí několik různých komunikačních rozhraní. Nástroj ForensIRT využívá REST API ke komunikaci s API serverem Cuckoo, který běží společně s hlavním serverem Cuckoo, jež obstarává správu virtuálních strojů a analýzu zaslaných vzorků malware. Při zaslání vzorku malware se nástroj ForensIRT pravidelně pomocí REST API dotazuje API serveru Cuckoo zda je již analýza dokončena a jakmile je, z výsledků analýzy poté moduly ForensIRT vytvoří webovou technickou zprávu (resp. její část).

Nevýhodou přidání dynamické analýzy do analyzačního procesu nástroje ForensIRT však jednoznačně je navýšení doby trvání analýzy malware. Zatímco statická analýza malware vždy trvala jen jednotky sekund, v případě zapojení dynamické analýzy malware je třeba přičíst dobu jenž drvá analýza malware nástrojem Cuckoo. Tedy především dobu po jakou necháme vzorek malware běžet a pozorujeme jeho chování. Pokud by byla tato doba příliš krátká, mohli bychom ukončit analýzu dříve než malware provede nějakou důležitou škodlivou činnost a mohli bychom pak falešně identifikovat vzorek jako neškodný. Naopak příliš dlouhá doba by nadměrně zvyšovala dobu analýzy a analýza nástrojem ForensIRT by poté byla velmi zdlouhavá a uživatelsky nepřívětivá. Tato doba lze libovolně měnit v konfiguračních souborech nástroje Cuckoo a v době testování nástroje ForensIRT byla nastavena nejdříve na 30, později na 20 sekund (snížení na 20 sekund nevedlo k žádnému znatelnému zhoršení analýzy, ovšem bylo by vhodné v tomto směru provést další důkladné měření a

[4]Dostupná z `https://github.com/Yara-Rules/rules`

srovnání před dalším snížením na méně než 20 sekund). Kromě času samotné analýzy je však také třeba přičíst čas spojený se správou virtuálních strojů. Především je třeba identifikovat volný virtuální stroj, sestavit čistou instanci (existující použitá instance totiž může být infikována dříve zkoumaným vzorkem a analýza na takovém stroji by mohla mít nesprávné výsledky), spustit nově sestavenou čistou instanci a do ní nahrát přes síť vzorek malware jež bude analyzován. Dohromady tedy přidání dynamické analýzy do procesu prodlouží dobu analýzy vzorku malware nástrojem ForensIRT až mnohonásobně, na celkovou dobu několika desítek sekund.

6.3 Implementační detaily

Jak bylo popsáno již v kapitole 6.2.1, nástroj ForensIRT, vzhledem ke skutečnosti že jeho výstupem je technická zpráva ve formátu HTML a proto využívá framework Django, je založen na návrhovém vzoru `Model Template View` (`MTV`). Proto se nyní podívejme na rozložení souborů nástroje ForensIRT (vyjmenovány budou pouze ty nejdůležitější):

- `/README.md` – Soubor dokumentující instalaci a základní použití nástroje.
- `/manage.py` – Vstupní bod aplikace, implementován a poskytován nástrojem Django.
- `/forensirt/` – Složka obsahující konfiguraci webové služby ForensIRT, tedy především konfigurační soubory pro serverovou část poskytovanou nástrojem Django.
- `/malware_analysis/` – Složka obsahující implementaci nástroje ForensIRT jako modulu pro webovou aplikaci Django. V této složce jsou všechny soubory spojené s funčností nástroje ForensIRT.
- `/malware_analysis/urls.py` – Soubor obsahující schéma dekódování `URL` adres (tedy seznam vzorů adres a příslušné třídy a jejich metody, jež budou volány při přístupu k těmto adresám).
- `/malware_analysis/views.py` – Soubor obsahující `View` komponentu návrhového vzoru `MTV`, tedy logiku zajišťující vykreslení výstupu aplikace na základě dat získaných z `Modelu` a s využitím šablon `Template`.
- `/malware_analysis/models.py` – Soubor obsahující `Model` komponentu návrhového vzoru `MTV`. Tento soubor obsahuje hlavní kontrolér zajišťující spouštění jednotlivých analyzačních modulů a tedy jádro aplikace. Na tento soubor se podíváme podrobněji v následující podkapitole.
- `/malware_analysis/config.cfg` – Soubor obsahující konfiguraci nástroje ForensIRT a jeho analyzačních modulů. Všechny hodnoty jsou ve výchozím stavu předkonfigurovány na vhodné produkční hodnoty, avšak jsou jednotně umístěny v tomto souboru, aby byla umožněna jednoduchá a centrální správa významných proměnných napříč nástrojem ForensIRT.
- `/malware_analysis/analysis_modules/` – Složka obsahující implementaci jednotlivých analyzačních modulů. Každý analyzační modul se zde nachází ve svém vlastním souboru.

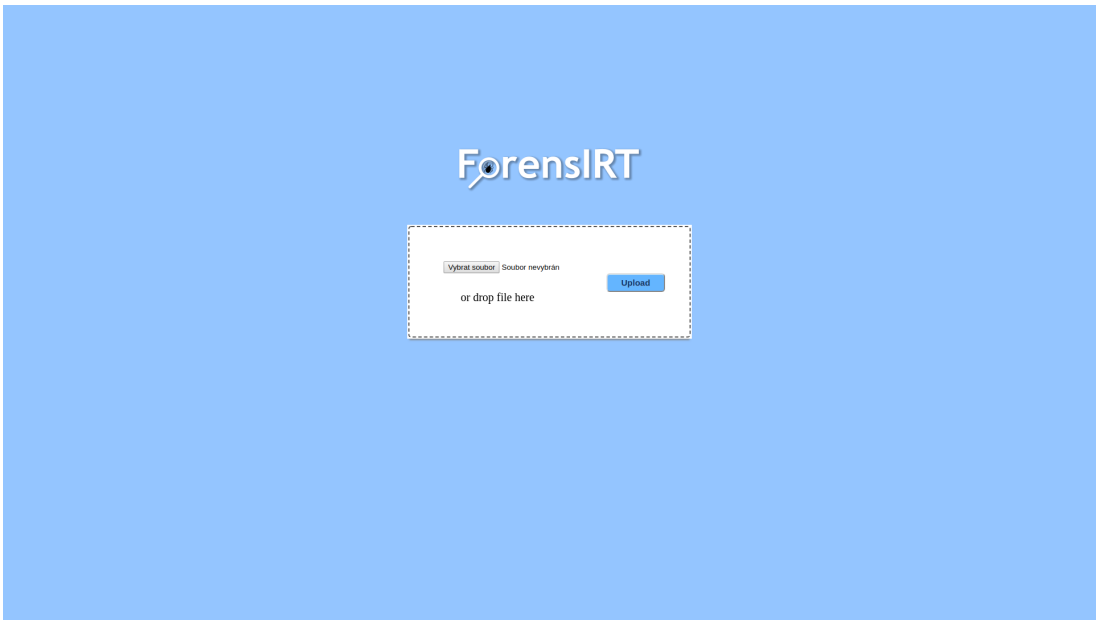
- `/malware_analysis/analysis_modules/__init__.py` – Soubor obsahující definici aktivních modulů – při instalaci nového modulu resp. k deaktivaci nechtěného existujícího modulu je třeba jej zde přidat, resp. zakomentovat.
- `/malware_analysis/analysis_modules/analysis_module.py` – Soubor obsahující definici třídy rodiče všech analyzačních modulů. Na tento soubor se dále podíváme v podkapitole 6.3.2.
- `/malware_analysis/static/` – Složka obsahující statické soubory potřebné k vykreslení webového výstupu, jako např. šablony CSS, ikony, skripty v jazyce Javascript apod.
- `/malware_analysis/templates/malware_analysis/` – Složka obsahující šablony k vykreslení webové aplikace.
- `/malware_analysis/templates/malware_analysis/analysis_modules/` – Složka obsahující šablony k vykreslení jednotlivých analyzačních modulů. Každý soubor dle svého názvu odpovídá jednomu analyzačnímu modulu a definuje přesný způsob jakým budou vykresleny informace získané analyzačním modulem.
- `/malware_analysis/templatetags/` – Složka obsahujcíí soubory s definicí příkazů k použití v šablonách pro usnadnění vykreslení obsahu, resp. rozšíření možností šablonovacího systému Django.
- `/malware_analysis/yara-rules/` – Složka s repozitářem pravidel YARA jak byly popsány v kapitole 6.2.3.

6.3.1 Kostra aplikace

Úvodní stránkou aplikace je formulář k nahrání vzorku k analýze. Po nahrání vzorku buďto stisknutím tlačítka `Zvolit soubor...`, popř. přetáhnutím souboru z počítače do vyznačené oblasti (tzv. drag-n-drop) a následně potvrzením stisknutí tlačítka `Upload` se soubor přenese pomocí protokolu `HTTP` metodou `POST` na webový server Django, odkud je předán do `View`, který zajistí vytvoření nové vlastní složky v úložišti analyzovaných vzorků malware (přesné umístění tohoto úložiště je snadno konfigurovatelné v souboru `/malware_analysis/config.cfg` avšak ve výchozím stavu je tato proměnná nastavena na hodnotu `/usr/share/forensirt/malware`) a přesunutí tohoto nově nahraného vzorku do této nové vlastní složky. Následně `View` provede přesměrování uživatele na unikátní adresu identifikující tento právě nahraný vzorek malware. Pokaždé když někdo navštíví tuto adresu, vyhledá se v úložišti malware příslušný vzorek malware a provede se opět nová analýza (takže v budoucnu není třeba znovu nahrávat vzorek malware, stačí znát unikátní adresu `URL` pro tento vzorek). Tato funkcionalita by se však dala snadno vypnout, v případě že by byla nežádaná – je totiž plně implementováno kompletní ukládání dokončené analýzy malware a lze tedy místo opětovné analýzy již jednou analyzovaného vzorku pouze zobrazit tuto starou (avšak většinou stále aktuální) technickou zprávu. Ve výchozím stavu je však tato funkcionalita vypnuta, tedy přestože se výsledná zpráva ukládá, vždy je při obnovení stránky vygenerovaná nová technická zpráva a ta stará je zahozena, především z důvodu usnadnění a urychlení vývoje a testování aplikace.

Jak již bylo zmíněno, každý vzorek nahraný do systému má unikátní adresu, pomocí nějž lze vzorek jednoznačně identifikovat a přistoupit k němu. Tato adresa je také využita k vytvoření vlastního adresáře v úložišti vzorků malware. Protože požadavkem na implementaci

Obrázek 6.3: Snímek úvodní stránky nástroje ForensIRT umožňující nahrání vzorku malware k analýze



je mít pro každý vzorek malware vlastní složku, ve které jsou ukládány soubory související s analýzou daného vzorku, pak zřejmě je třeba vhodně zvolit název této složky, která zároveň bude sloužit jako unikátní identifikátor daného vzorku malware (jde o zřejmý důsledek požadavku vlastní složky pro každý vzorek malware). Je očividné, že nelze vycházet z názvu souboru vzorku malware, především protože název souboru je snadno modifikovatelný, mohlo by tedy dojít k situaci kdy v databázi jsou dva totožné vzorky malware ovšem oba pod jiným názvem, což porušuje podmínku unikátního identifikátoru pro každý unikátní vzorek malware. Navíc autoři malware často v rámci strategie ukrytí před odhalením volí jako název daného vzorku názvy běžných souborů, jako například `system32.dll`, `Skype.exe` a podobně. Proto by se tedy mohlo snadno stát, že dva různé vzorky malware by měly shodný identifikátor.

Vzorek malware však rozlišuje kontrolní součet, kryptografická hash, či jiná podobná matematická operace nad samotným vzorkem malware (tedy operace nad binárním souborem daného vzorku). Bylo rozhodnuto, že podobně jako např. u služby VirusTotal, popsané dříve v kapitole 3.2.1, bude použita kryptografická hash binárního souboru vzorku malware. Ovšem kryptografické hašovací funkce jsou v oblasti bezpečnosti velmi žhavé téma. Neustále jsou objevovány nové zranitelnosti a nedostatky v kryptografických funkcích, jako např. nechvalně známé a často ještě velmi používané `MD5` či `SHA-1`. Proto bylo rozhodnuto využít ne jedinou hašovací funkci, ale rovnou několik (ve výchozím nastavení dvě), jež jsou za sebe zřetězeny (v případě identifikátoru jsou odděleny lomítkem a v případě úložiště malware se tedy jedná o vnořené složky). Navíc nástroj ForensIRT je implementován tak, že volba konkrétních hašovacích funkcí (případně jejich počet) je ponechána na uživateli nástroje. Díky tomu je snadné v budoucnu změnit schéma adresování vzorků malware, v případě že by byl některý z použitých algoritmů uznán za nevhodný. Použití dvou (či více) hašovacích funkcí nepřináší žádné nevýhody, jediné omezení je maximální povolená délka adresy `URL`, která při obsáhnutí unikátního identifikátoru vzorku malware nesmí přesáhnout svou pevně stanovenou hranici (přibližně dva tisíce znaků), a podobně souborové systémy

požadují aby všechny cesty k souborům v daném souborovém systému nepřesahovaly příslušnou maximální přípustnou délku. Mimo tyto horní hranice délky identifikátoru vzorku malware však jedinou nevýhodou může být uživatelská přívětivost. Ovšem i v případě využití 2 hashovacích funkcí o délce výstupu 1024 bitů se v kódování o bázi 16 bitů jedná o dva řetězce o délce 64 znaků, tedy mnohonásobně nižší než zmíněné hranice nastavené délkou `URL` či souborovým systémem a stále uživatelsky přívětivé (bez problému zobrazitelné na jediném řádku při použití běžně dostupných moderních displejů s vysokým zobrazovacím rozlišením).

Po otevření adresy obsahující unikátní identifikátor vzorku malware se zavolá metoda `generate_report` v souboru `views.py`, jež zajistí vyhledání odpovídajícího vzorku malware v lokálním úložišti malware a poté spuštění analýzy a následné vykreslení výstupu v podobě technické zprávy. Navíc je zajištěno uložení technické zprávy uzpůsobené pro samostatné prohlížení bez nutnosti připojení k aplikaci. Tato varianta je totáž technická zpráva, ovšem bez prvků jež dávají smysl pouze v on-line verzi nástroje - tedy především tlačítka v záhlaví sloužící ke stažení off-line archivu a zadání dalšího vzorku k analýze - k oběma těmto akcím je třeba mít přístup k nástroji a nejsou tedy v off-line verzi k dispozici). Tato verze je poté spolu se všemi ostatními soubory získané v průběhu analýzy zabalena do archivu, jež může být operátorem nástroje stažen. Tento archiv poté kromě této speciální technické zprávy obsahuje také kompletní detailní technické výpisy z analýzy (např. kompletní výpis formátu PE, kompletní zpráva dynamické analýzy nástrojem Cuckoo ve formátu `json` apod.) a další relevantní soubory. Tato funkčnost byla přidána především z toho důvodu, že v oblasti bezpečnosti a především při analýze malware a v bezpečnostních týmech CSIRT je kritickým aspektem další šíření informací ať již v rámci týmu či komunikace na státní či mezistátní úrovni s ostatními bezpečnostními týmy s cílem sdílet znalosti získané o útočnících a jejich postupech a nástrojích. Tato funkce co nejvíce usnadňuje právě toto sdílení informací.

Hlavním úkolem celého nástroje je analýza vzorku. Správu procesu analýzy obstarává třída `SampleAnalysis` definovaná v souboru `models.py`. Tato třída, jak bylo již dříve řečeno v předchozí kapitole, nahrazuje standardní třídu `Model` v návrhovém vzoru `MTV`. Ve `View` je vytvořena nové instance této třídy a následně je nad touto instancí provedena operace `analyze`, jež obstará analýzu všemi aktivními analyzačními metodami. Novou instanci třídy `SampleAnalysis` lze kontruovat dvoumi způsoby. První možnost je konstruktoru třídy předat parametr se jménem `file`, jež obsahuje jako hodnotu cestu k souboru vzorku, jež budeme analyzovat. V našem případě adresu k souboru jež byl právě uživatelem nástroje nahrán do systému. Při této variantě konstrukce třídy `SampleAnalysis` je vzorek nahrán do úložiště vzorků malware a odtud se s ním dále pracuje. Tento způsob tedy umožňuje zavedení nového vzorku malware do úložiště vzorků. Druhý způsob konstrukce očekává jako parametr konstruktoru `hash`, jehož hodnota bude nastavena na kryptografické hashe vzorku malware. Pomocí těchto hashí bude vzorek vyhledán v databázi. Díky této druhé metodě lze opětovně pracovat se vzorky jež již v databázi jsou uloženy z důvodu předchozí analýzy daného vzorku.

Konečně metoda `analyze` třídy `SampleAnalysis` bude postupně spouštět aktivní analyzační metody. Protože analyzační metody mají definované závislosti (tedy jedna analyzační metoda může záviset na jiné, např. protože k analýze využívá také data jež byla získána první metodou), mohlo by dojít k uváznutí kdy jsou například dvě metody navzájem na sobě závislé (tedy první metoda má jako prerekvizitu druhou metodu a obráceně). Proto algoritmus v cyklu spouští stále nedokončené analyzační metody a pozoruje zda některá z nich dokončila analýzu. Jestliže v některé iteraci dojde k tomu, že žádné moduly nebyly schopny dokončit svou analýzu (ať již proto že všechny moduly již analýzu skončily, nebo

všechny moduly čekají na dokončení jiných modulů), je běh ukončen. Algoritmus ilustruje následující pseudo-kód:

```
array pendingMethods;
int finishedThisRound;

do {
    finishedThisRound = 0;
    for each method in pendingMethods {
        method.analyze(malwareSample);

        if (method.finished == True) {
            pendingMethods.pop(method);
            finishedThisRound++;
        }
    }
} while (finishedThisRound > 0);
```

Jestliže na konci tohoto úseku pseudo-kódu je množina `pendingMethods` neprázdná, nastala situace kdy některé z metod jsou na sobě navzájem závislé. Na tuto skutečnost bude uživatel upozorněn varováním v technické zprávě. Počet provedených cyklů tohoto algoritmu se bude rovnat jedné v případě kdy žádné z metod na sobě nejsou nijak závislé a v nejhorším případě se cyklus bude opakovat `(N + 1)`-krát kde `N` je počet analyzačních metod, a to v případě kdy všechny metody jsou na sebe postupně závislé (jestliže relaci závislosti metody A na metodě B zapisujeme jako `A` $\rightarrow$ `B`, pak by šlo o situaci `A` $\rightarrow$ `B` $\rightarrow$ `C` a tedy `N = 3` a počet průchodů cyklem by byl roven číslu 4). Předchozí tvrzení je horní omezení a platí i pro případy kdy jsou některé moduly ve vzájemné závislosti (a tedy například pro `A` $\rightarrow$ `B` $\rightarrow$ `C` $\rightarrow$ `A` se cyklus provede jedinou a poté bude algoritmus ukončen s množinou `pendingMethods = {A, B, C}`).

6.3.2 Analyzační moduly

Jednotlivé analyzační moduly, jež jsou spouštěny v rámci analýzy popsané v předchozí kapitole, jsou potomky třídy `AnalysisModule` definované v souboru `analysis_module.py`. Nejdříve se v detailu podíváme na tuto rodičovskou třídu a poté prozkoumáme také jeden modul jako ukázku implementace vlastního modulu, jakožto hlavního stavebního bloku tohoto nástroje.

Třída `AnalysisModule` má následující strukturu:

- `class AnalysisModule extends ABC`
  - `attribute prerequisite_modules`
  - `abstract method analyze(sample, knowledgebase)`
  - `class method is_ready(knowledgebase)`
  - `static method is_finished(module, knowledgebase)`

V záhlaví souboru definujícího třídu `AnalysisModule` nalezneme následující kód:

```
from abc import ABC, abstractmethod
class AnalysisModule(ABC):
    prerequisite_modules = []
```

Za povšimnutí stojí především definice třídy, tedy že je potomkem `ABC`, tedy `Abstract Base Classes`, třídou užívanou k definicí abstraktních tříd. Jediným atributem nacházejícím se ve všem potomcích je poté definován atribut `prerequisite_modules`, jež definuje seznam (`list`) řetězců obsahující názvy analyzačních metod jež jsou prerekvizity pro danou analyzační metodu.

```
@abstractmethod
def analyze(sample, knowledgebase):
        pass
```

Dále následuje definice abstraktní metody `analyze`. Tato metoda přijímá dva argumenty a to `sample` jež je řetězec obsahující absolutní cestu k analyzovanému vzorku malware a `knowledgebase` jež je slovník (`dict`) obsahující všechny informace již dokončených analyzačních modulů (tedy také zcela určitě modulů definovaných v seznamu `prerequisite_modules`) každé pod klíčem názvu modulu, a navíc pod klíčem `config` také hodnoty v konfiguračním souboru `config.cfg`. s tímto kontextem mohou potomci nakládat dle uvážení (samozřejmě by neměli měnit obsah znalostní databáze `knowledgebase` či měnit samotný binární soubor malware) a na výstupu metody je pak očekáván slovník (`dict`) obsahující výstup analyzačního modulu, o nějž bude následně rozšířena znalostní databáze `knowledgebase`.

```
@classmethod
def is_ready(cls, knowledgebase):
        for prereq in cls.prerequisite_modules:
                if not (prereq in knowledgebase):
                        return False
        return True
```

Následuje definice metody `is_ready`, jež kontroluje zda je tato analyzační metoda připravena k analýze, tedy především zda všechny metody definované v prerekvizitách této metody (`prerequisite_modules`) již proběhly a jejich výstup je uložen ve znalostní databázi `knowledgebase`. Tato metoda je použita k vyhodnocení zda je možno spustit analyzační metodu v algoritmu popsaném v přechozí kapitole.

```
@staticmethod
def is_finished(module, knowledgebase):
        return module in knowledgebase \\
            and (not ('finished' in knowledgebase[module]) \\
            or knowledgebase[module]['finished'] == True)
```

Analyzační metoda může skončit svůj běh avšak výsledek může být selhání (například v případě kdy nelze na daný vzorek aplikovat danou analyzační metodu – například pokud analyzovaný vzorek malware je infikovaný textový dokument pro spuštění aplikací Microsoft Word, pak zřejmě nelze provést analýzu formátu PE, protože daný vzorek není tohoto formátu). Proto je třeba předat tuto informaci dál a to tak, že ve výstupu dané analyzační metody se objeví pod klíčem `finished` hodnota `False`. V případě že je `finished` nastaveno na hodnotu True, pak je očekáváno že modul proběhl v pořádku a ostatní moduly jež jej mají jako prerekvizitu mohou s jeho daty v pořádku pracovat. V případě kdy je `finished` nedefinováno, předpokládá se že modul doběhl v pořádku (tedy je lze presumpce úspěšného dokončení, především z důvodu zkrácení a zpřehlednění kódu analyzačních). Ke kontrole tohoto stavu slouží právě metoda `is_finished`.

Nyní si na analyzační metodě `strings_module` ukažme implementaci analyzačního modulu. Analyzační metoda `strings_module` slouží k extrakci a zobrazení binárních řetězců ze souboru analyzovaného vzorku, tedy jde o implementaci analyzační metody popsané v kapitole 3.1.1.

```
class StringsModule(AnalysisModule):
    prerequisite_modules = []

    encodings = {'ascii': 1, 'utf-16': 2, 'utf-32': 4}
    min_length = 3
```

Zřejmě prerekvizity tohoto modulu nejsou žádné, protože k získání řetězců ze souboru vzorku není třeba mít žádné předchozí informace než jsou již poskytnuty – tedy cestu k souboru vzorku, jež je poskytnuta metodě `analyze` v parametru `sample`, jak bylo popsáno výše. Následně si definujeme dva atributy a to encodings jež je slovník (`dict`) obsahující jako klíče názvy podporovaných kódování (tedy kódování jež se budeme snažit ze vzorku získat) a jako hodnotu velikost jednoho znaku daného kódování v bajtech. Konečně druhý vlastní atribut `min_length` definuje minimální délku řetězce, tedy kolik znaků po sobě jdoucích musí být v daném kódování tisknutelné znaky.

```
@classmethod
def analyze(cls, sample, knowledgebase):
        return_data = {}
        for enc, bytelen in cls.encodings.items():
                return_data[enc] = cls.extract_strings(sample, enc, bytelen)
        return {'strings': return_data}
```

Konečně metoda `analyze` postupně projde všechna zadaná kódování a zavolá funkci `extract_strings`, jež ze souboru analyzovaného vzorku získá řetězce daného kódování o stanovené minimální délce a výsledek uloží do slovníku v proměnné `return_data`, jež vrátí příkazem `return`. Metody vycházející z této metody poté mohou k těmto datům přistoupit z proměnné `knowledgebase['strings_module']['strings']`, např. pro získání všech řetězců nacházejícím se v souboru vzorku malware v kódování UTF-8 přístupem k `knowledgebase['strings_module']['strings']['utf-8']`). Metodu `extract_strings` zde rozebírat nebudeme. Podobným způsobem jsou také implementovány ostatní moduly, proto je zde také rozebírat nebudeme.

Posledním důležitým aspektem modulů je jejich vzhled v technické zprávě, jež je výstupem analýzy vzorku. Tyto jsou definovány ve vlastních souborech ve formátu `HTML`. Využit je šablonovací systém poskytovaný frameworkem Django. K proměnným lze tedy přistoupit pomocí speciálního jazyka. Protože design výstupu modulu `strings_module` popisovaného výše je velmi komplikovaný a rozsáhlý (především protože výstup vykresluje do interaktivní tabulky za použití knihovny KingTables a dále implementuje vlastní skripty v jazyce JavaScript pro práci s touto tabulkou – především protože knihovna KingTables ve výchozím stavu neposkytovala požadované funkce) alespoň si na příkladu ukážeme jak by taková šablona pro modul `strings_module` vypadat ve velmi zjednodušeném případě:

```
{% load questionmark_icon %}
<h1>Binary Strings {{ "Module description here"|questionmark_icon }}</h1>
<div class="box_content">
    <p>
                {% for encoding, found_strings in strings.items %}
```

```
                <h2>{{ encoding }}</h2>
                {% for the_string in strings.items %}
                {{ the_string.3 }}<br/>
                {% endfor %}
                {% endfor %}
    </p>
</div>
```

Na prvním řádku lze vidět načtení vlastního příkazu (tzv. `tagu`) `questionmark_icon`, jež slouží k vykreslení ikony otazníku jež zobrazí při najetí myši bublinu s popisem jež je předán tomuto příkazu v argumentu. Ihned na dalším řádku je tento příkaz použit, aby do záhlaví modulu obsahující název modulu přidal právě takovou vysvětlivku, jež umožní uživateli zobrazit podrobnější popis modulu než jen jeho název. Následuje obsah modulu, jež je jednoduchý cyklus procházející všechna kódování a všechny řetězce v těchto kódování a zajiťuje jejich výpis, každý řetězec na svém řádku (`the_string` je proměnná jež obsahuje pole kde na 3. pozici v tomto poli je uložen získaný řetězec, proto je v kódu `the_string.3`). Kód je velmi snadno pochopitelný sám o sobě a proto je netřeba ho dále rozebírat.

Podobným způsobem jsou implementovány i další analyzační moduly. Je zřejmé že implementace modulárnosti nástroje je velmi silná a umožňuje bez jakéhokoliv omezení vývoj samostatných modulů. Kód a vzhled každého modulu je ve svém vlastním souboru a systém je implementován tak, aby tyto soubory obsahovaly čistý a přehledný kód týkající se pouze činnosti daného modulu a nebyl tedy tento kód zatěžován zbytečnou aplikační logikou. Na příkladu byla demonstrována elegance a síla tohoto systému. Nebyla věnována pozornost implementaci samotných analyzačních metod, protože ty jsou navzájem velmi rozdílné a bylo by nad rámec této práce dopodrobna vysvětlovat jejich implementaci. Navíc jejich implementace vychází z poznatků popsaných v kapitole 2.

6.4 Výkonnostní test

Na závěr této kapitoly se podívejme na výkonnost nástroje ForensIRT, tedy rychlost jakou dokáže analyzovat vzorky malware a zobrazit vyšetřovateli technickou zprávu.

6.4.1 Metodologie

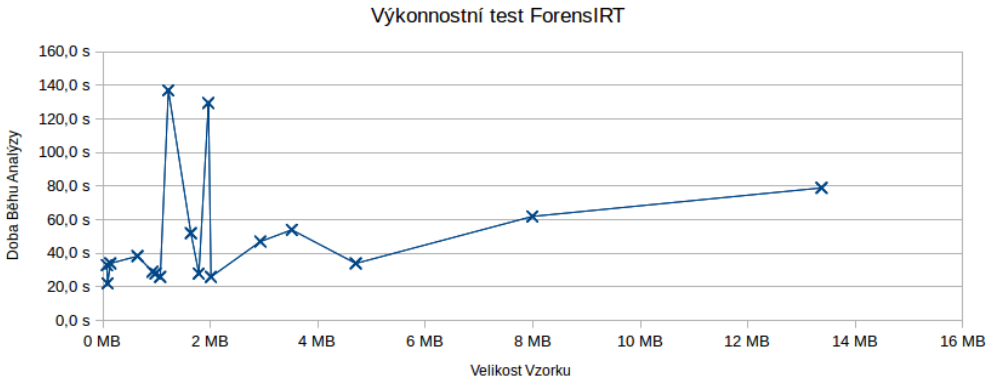
Cílem tohoto testu bylo změřit rychlost analýzy v závislosti na velikosti zkoumaného vzorku. Zdrojem k testování byla veřejně dostupná a frekventovaně aktualizovaná databáze vzorků malware s názvem theZoo[5]. Z této databáze bylo vybráno 16 vzorků malware. Množina testovaných vzorků byla navc doplněna o 3 další běžné programy (tedy bezpečné neškodlivé programy), aby bylo otestováno zda nástroj pracuje se všemi programy stejně dlouho, nezávisle na jejich škodlivosti. Při výběru byl brán ohled především na cílovou platformu (muselo se jednat o malware pro operační systémy Windows) a poté na velikost vzorku malware – byly vybrány největší vzorky malware jež se v této databázi nacházely. Tyto vzorky byly poté postupně podrobeny opakované analýze nástrojem ForensIRT. Časy jednotlivých vzorků byly zprůměrovány a výsledné časy byly zaneseny do grafu.

[5]Dostupná z `https://github.com/ytisf/theZoo`

6.4.2 Výsledky testu

Výsledky testu lze shlédnout v obrázku 6.4. Testované vzorky malware byly v rozsahu `68 KB` až `13,4 MB`. Doba běhu se pohybovala od 19,8 sekundy až po 137 sekund. V grafu jsou zřetelně rozdílné (nadměrně vysoké) doby běhu analýzy u dvou vzorků o velikostech `1,22 MB` a `1,96 MB`. Po důkladném šetření bylo zjištěno, že důvodem pro tyto anomálně nadměrné výsledky bylo u těchto vzorků zpracování výsledků dynamické analýzy. Tyto vzorky malware totiž oproti ostatním vzorkům vykonaly nadměrné množství aktivity v průběhu jejich analýzy. Jeden ze vzorků byl například malware typu ransomware, jež během 20 sekund svého běhu začal šifrovat a zamykat tisíce souborů na virtuálním stroji kde byl analyzován. Tato činnost generovala velké množství systémových volání pro přístupy k souborům aj. Z toho důvodu výsledná zpráva dynamické analýzy nástrojem Cuckoo, jež byla zdrojem informací pro nástroj ForensIRT, byla nadměrně velká a její zpracování trvalo nadměrně dlouhou dobu. Tato skutečnost by se dala potlačit upravením parametrů výstupní zprávy nástroje Cuckoo, v případě že by bylo žádané i pro tyto vzorky malware zkrátit dobu analýzy.

Obrázek 6.4: Test rychlosti analýzy nástrojem ForensIRT v závislosti na velikosti vzorku



Jestliže opomeneme tyto dva anomální vzorky, pak je z výsledků zřejmé, že doba analýzy vzorku malware je přibližně lineárně závislá na jeho velikosti, kdy pro největší vzorek malware (`13,4 MB`) byla doba běhu 79 sekund. V užité databázi vzorků malware se však průměrně velikost vzorku malware pohybovala v řádu pár stovek kilobajtů, kde průměrná doba běhu analýzy vzorků o takové velikosti byla v řádu pár desítek sekund.

Kapitola 7

Analýza vzorku malware

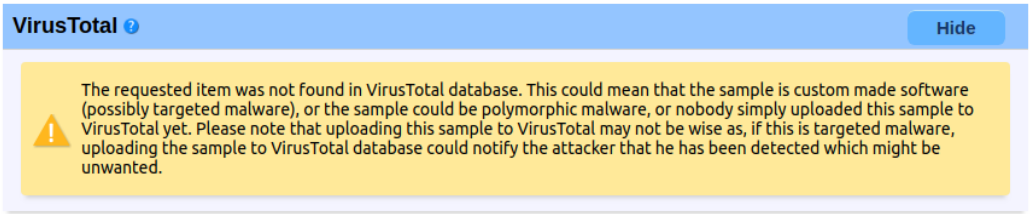
7.1 Analýza nástrojem ForensIRT

Předmětem analýzy je vzorek s následujícími parametry:

- Název souboru: `Skype.exe`
- Velikost souboru: `77 824 Bajtů`
- MD5 Hash: `505b1727cbf13f96fe914daba73a808d`
- SHA-256 Hash: `aca90947016e7e035deac1e0588cc8aca51600b2f7e22ca0ce0e0a4d92233e63`

Analýza modulem VirusTotal (viz obrázek 7.1) skončila neúspěchem, tedy vzorek s danou kryptografickou haší nebyl doposud nikdy do nástroje VirusTotal nahrán. Tato skutečnost může znamenat že se jedná o obskurní aplikaci, jež doposud nikdo nenahrál do tohoto nástroje k analýze, nebo se může jednat o jedinečný vzorek malware užitý k cílenému útoku. Každopádně se jedná o dosud neviděný a tedy běžně neužívaný program.

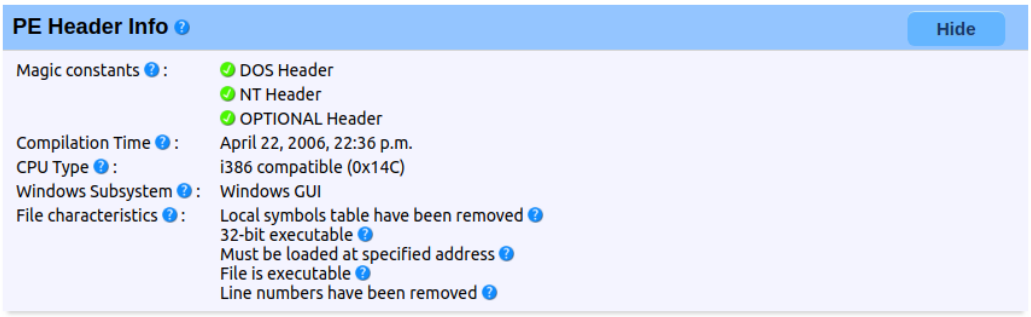
Obrázek 7.1: Výsledky analýzy modulu Virus Total



Analýza modulem PE Header Info (viz obrázek 7.2) potvrzuje skutečnost že se jedná o spustitelný soubor pro platformu Windows na 32bit kompatibilní architektuře. Doba kompilace je v záhlaví vzorku nastavena na datum 22. Dubna 2006 ve 22:36 hodin. Tato informace je obzvlášť udivující v souvislosti s výsledky modulu VirusTotal – je poměrně překvapující že by 12 let starý program nebyl nikdy nahrán do nástroje VirusTotal. Z této informace tedy můžeme vyvodit že datum kompilace je buď podvržené, nebo se jedná o obecně nepoužívaný, neveřejný program. Program je navržen pro užití grafického rozhraní Windows, nejedná se tedy o konzolovou aplikaci. Navíc má program příznak zajišťující že bude při spuštění nahrán na specifickou adresu v paměti, což může být případně vzorkem

vyžito k celé řadě útoků. Navíc nebyly v záhlaví přidány žádné textové informace o daném programu (jako např. název a verze programu, vydavatel apod.), což bývá typické u běžných programů, jako například program Skype, o nějž by se podle názvu programu mělo jednat.

Obrázek 7.2: Výsledky analýzy modulu PE Header Info

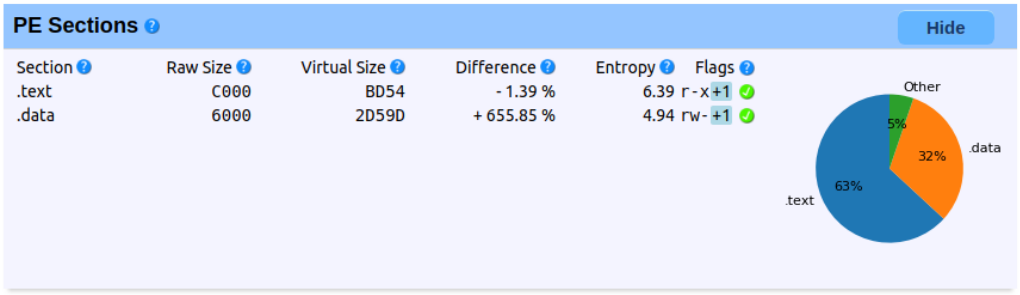


Analýza modulem PE Import Directory (viz obrázek 7.3) je zřejmé, že vzorek načítá knihovny za běhu (využívá tzv. runtime linking), protože jediná knihovna v Import Directory je `kernel32.dll` z níž je importována funkce `LoadLibraryA` jež bude dále vzorkem využita k importování dalších metod z dalších knihoven za jeho běhu. Ačkoliv tato metoda vede ke skrývání schopností vzorku při statické analýze, nelze spoléhat na škodlivost vzorku pouze na základě této skutečnosti, protože i běžné neškodlivé programy mohou tuto metodu využívat. Z analýzy tímto modulem tedy nelze vyvodit žádné důsledky.

Obrázek 7.3: Výsledky analýzy modulu PE Import Directory

Analýza modulem PE Sections poskytuje informace o sekcích daného vzorku. Je zřejmé že k dispozici jsou pouze dvě sekce, jedna obsahující kód aplikace a druhá obsahující data. Datová sekce je poměrně malá, avšak její velikost po alokování (`Virtual Size`) je skoro sedminásobná. Tato skutečnost by mohla znamenat že tento vzorek má schopnost získat nějaká další data za běhu, ať již rozbalením případných komprimovaných dat jež jsou v tomto spustitelném souboru přibalena, nebo třeba stažením dat ze sítě. Výsledky analýzy tímto modulem lze shlédnout na obrázku 7.4.

Obrázek 7.4: Výsledky analýzy modulu PE Sections

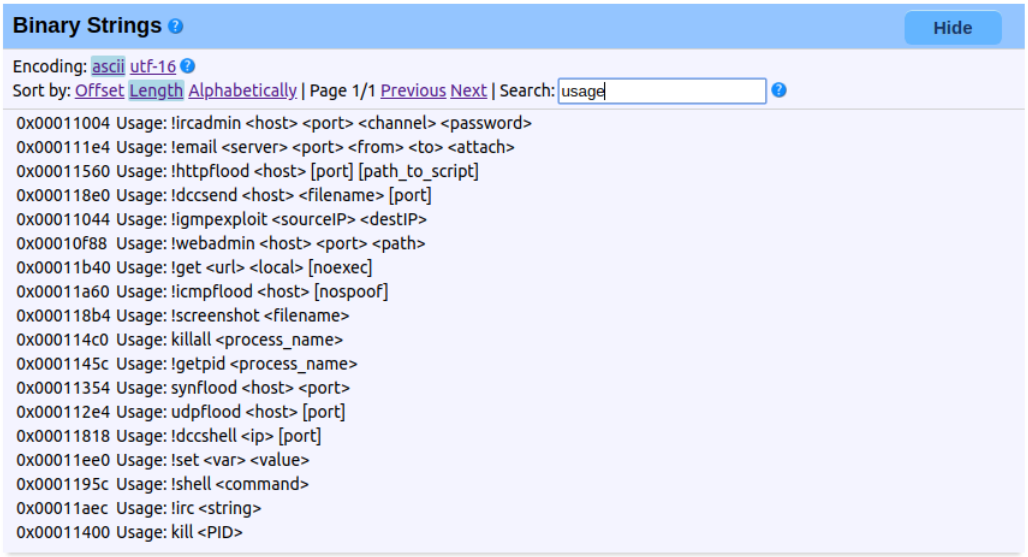


Podle výsledků modulu Strings vzorek obsahuje 30 řetězců v kódování UTF-16 jež se podle adres nachází na začátku datové sekce. Podle adres jednotlivých řetězců se zřejmě jedná o skupinu dvojic klíč-hodnota. Navíc se jedná o klíče jež typicky bývají v záhlaví spustitelného souboru PE, jako např. `ProductVersion` nebo `CompanyName`. Navíc skutečnost že se jedná o řetězce v kódování UTF-16 a že se nachází relativně na začátku datového oddílu napovídá že by v datovém oddílu mohl být umístěn další nešifrovaný program, jež bude pravděpodobně uložen na disk a spuštěn za běhu analyzovaného vzorku.

Dále analýza vzorku modulem Strings odhaluje celou řadu řetězců v kódování ascii. Po seřazení řetězců podle délky jsou ihned zřejmé zajímavé řetězce typu
`Mozilla/5.0 (Slurp/cat; vaginamook@inktomi.com;`
`http://www.supercocklol.com/slurp.html)`
a další podobné, jež jsou zřejmě řetězce využitelné jako hodnoty položky User-Agent v záhlaví HTTP hlaviček. Skutečnost že tento program obsahuje několik různých hodnot User-Agent je velmi podezřelá, protože tyto hodnoty jsou užívány k identifikaci programu jež přistupuje ke vzdáleným službám přes protokol HTTP. Využití více různých hodnot položky User-Agent tedy tento cíl potlačuje. Navíc obsah těchto hlaviček je také obzvlášť nezvyklý a napovídá že by se mohlo jednat o malware.

Dále mezi řetězci v kódování ASCII nalézáme řetězec
`Usage: !httpflood <host> [port] [path_to_script]`
jež je zřejmě nápověda k užití příkazu `httpflood`, což by dle názvu mohl být příkaz k aktivaci útoku HTTP Flood na daný cíl. Po vložení slova `usage` do vyhledávání řetězců v modulu Strings jsou získány všechny takové nápovědy k příkazům, kde lze nalézt další zřejmě škodlivé příkazy jako např. `shell`, `synflood`, `igmpexploit` a další. Na základě těchto znalostí se dá vyvodit, že se jedná s největší pravděpodobností o malware, pravděpodobně typu backdoor, jež dokáže přijímat příkazy tohoto typu. Veškeré známé příkazy s nápovědou lze shlédnout na obrázku 7.6.

Obrázek 7.5: Příkazy k útokům jež přijímá vzorek, získané modulem Strings

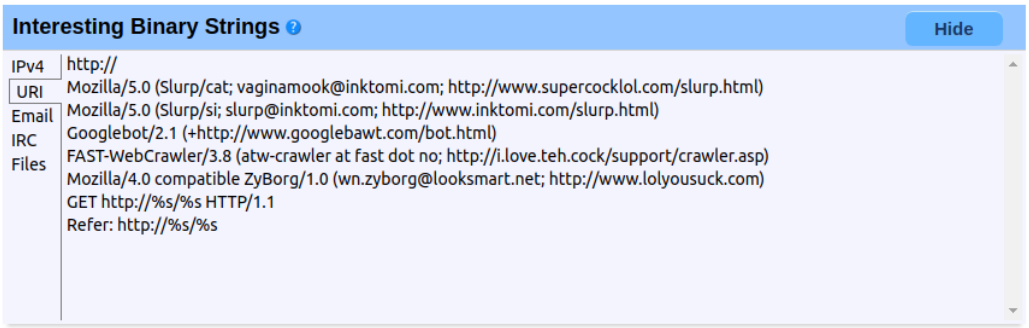


Modul Interesting Binary Strings (viz obrázek 7.6) dále poodhaluje další zajímavé řetězce. Především skutečnost že program obsahuje řetězce pro komunikaci textovým protokolem IRC, tedy za předpokladu že analyzovaný vzorek je skutečně malware, pak zřejmě

C2C server bude nějaký IRC server. V řetězcích nebyly nalezeny žádné IP adresy verze 4 ani 6. Naopak názvů souborů v řetězcích bylo mnoho, především:

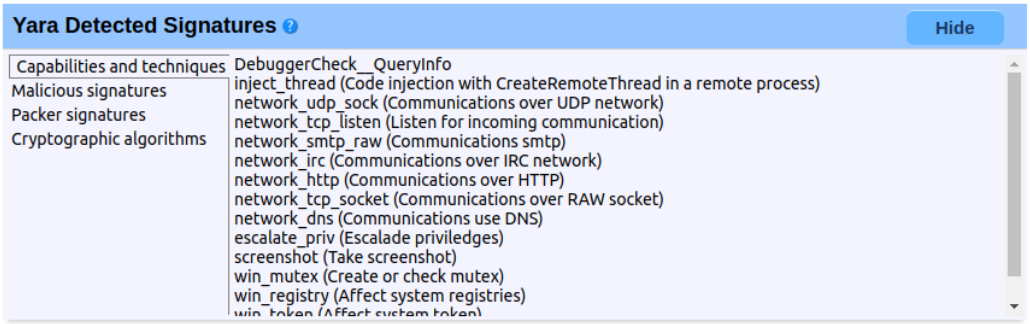
- `C:\WINDOWS\system32\drivers\ntndis.sys`
- `D:\D_DATA\i\evil\Driver\i386\Drv.pdb`
- `ntoskrnl.exe`
- `cmd.exe`
- `HAL.dll`
- `ndis32.sys`
- a řadu dalších souborů formátu PHP a PL, jako např. index.php apod.

Obrázek 7.6: Výsledky analýzy modulu Interesting Binary Strings



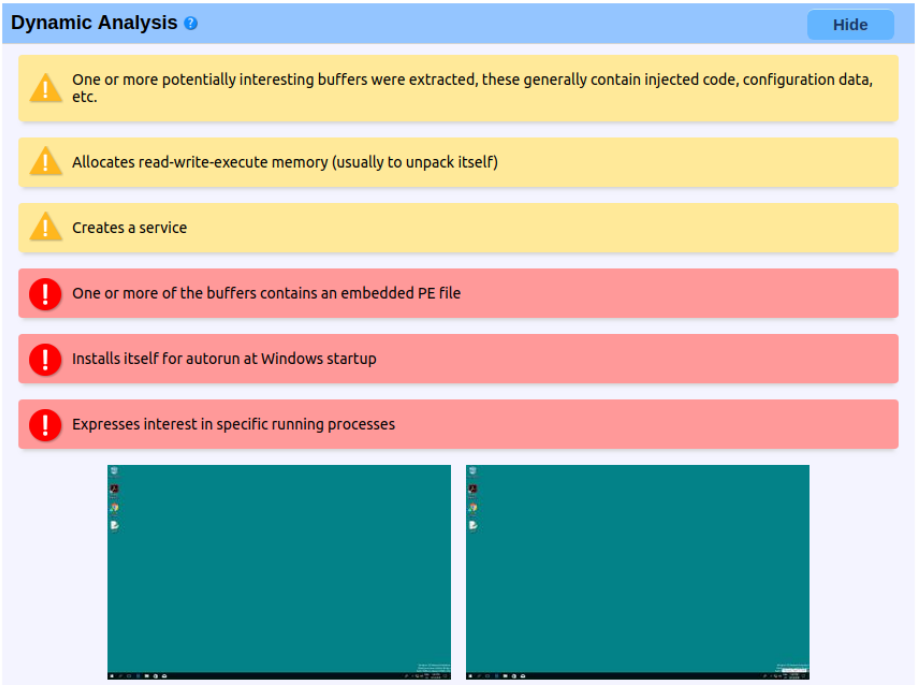
Modul pro analýzu pravidly YARA (viz obrázek 7.7) odhalil že program užívá komprimační algoritmus detekovaný pravidlem s názvem `Microsoft_Visual_Cpp_v50v60_MFC`. Dále ve vzorku byly identifikovány úseky kódu implementující kryptografickou hash funkci MD5 a kódovací a dekódovací funkce BASE64. Dále byly v kódu nalezeny signatury kódu pro přístup k internetu pomocí Windows Socket a schopnosti IRC komunikace, odesílání mailů pomocí protokolu SMTP, komunikaci přes TCP i UDP protokoly včetně poslouchání na portech daného stroje a konečně modifikaci registrů Windows a správu semaforů (angl. mutex). Tyto informace dále utvrzují podezření že se jedná o škodlivý kód a utvrzují dříve získané informace o schopnostech programu (z řetězců příkazů jež přijímá) a řetězců odpovídajících IRC komunikaci.

Obrázek 7.7: Výsledky analýzy modulu Yara Detected Signatures



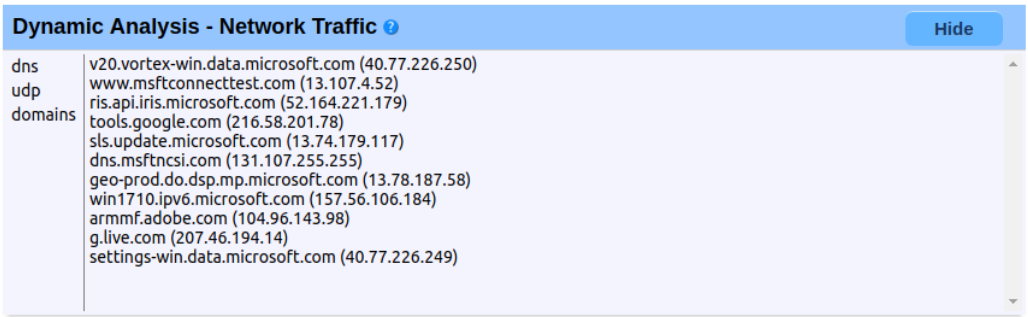
Dynamická analýza jejíž shrnutí je k nalezení v modulu Dynamic Analysis (viz obrázek 7.8) odhaluje že skutečně vzorek má v sobě přibalen další spustitelný soubor formátu PE (jak bylo již dříve předpokládáno z analýzy získaných řetězců). Navíc vzorek alokuje paměť se znaky pro zápis i spuštění, jež mu umožňuje nahrát do paměti počítače za běhu vlastní kód a spustit jej. Navíc vzorek vytváří systémovou službu, jež zřejmě zajistí spuštění programu při startu počítače. Snímky obrazovky z průběhu instalace ukazují pouze prázdnou pracovní plochu, je tedy zřejmé že ačkoliv vzorek v záhlaví PE má uvedeno že se jedná o aplikaci s grafickým rozhraním, žádné takové rozhraní se neukázalo, určitě tedy program není určen pro lidské operátory. Tyto skutečnosti jen utvrzují podezření na škodlivost analyzovaného vzorku.

Obrázek 7.8: Výsledky analýzy modulu Dynamic Analysis



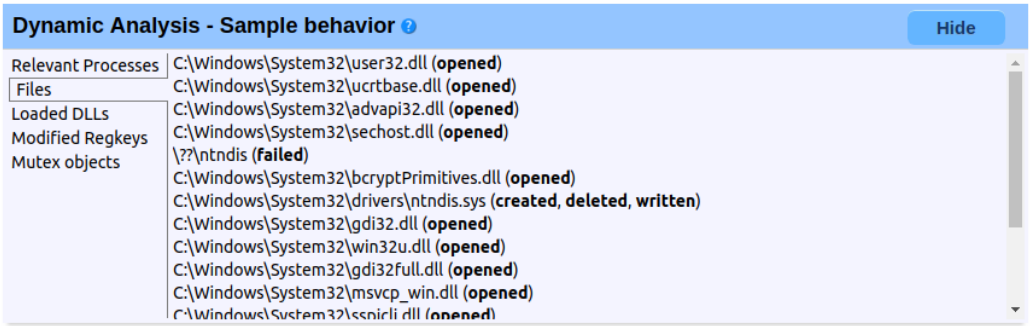
Síťová komunikace vzorku zaznamenaná v modulu Network Traffic (viz obrázek 7.9) obsahuje celou řadu DNS dotazů na zřejmě běžné domény. Vzorek se tedy zřejmě nesnaží kontaktovat žádné očividně podezřelé domény.

Obrázek 7.9: Výsledky analýzy modulu Dynamic Analysis - Network Traffic



Konečně z modulu Sample Behavior (viz obrázek 7.10) můžeme zjistit, že vzorek přistupoval k celé řadě systémových souborů a konečně do souboru `C:\Windows\System32\drivers\ntndis.sys`. Vzorek dále zapisoval do registrového klíče `HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node \Microsoft\Windows NT\CurrentVersion\Winlogon\Shell`, zřejmě pro zajištění spuštění vzorku po startu PC. Konečně vzorek vytvořil nový semafor se zdánlivě nesmyslným číselným označením, pravděpodobně kvůli zamezení opětovné infekce stejným vzorkem malware, což by mohlo vést k interferenci s již existujícím vzorkem.

Obrázek 7.10: Výsledky analýzy modulu Dynamic Analysis - Sample Behavior



7.1.1 Závěr analýzy

Z výsledků analýzy nástrojem ForensIRT je zcela zřejmé, že analyzovaný vzorek malware určitě není program Skype, jak by z názvu souboru mohlo vyplývat, ale jedná se o vysoce škodlivý vzorek malware, jež infikuje stroj na němž je spuštěný, zajistí automatické spuštění po startu, připojí se na vzdálený server a za pomocí protokolu IRC přijímá příkazy, jež umožňují spuštění libovolného kódu na infikovaném stroji, rozesílání nevyžádaných emailů a řadu dalších předdefinovaných útoků přes síť internet.

7.2 Porovnání výsledků

Vzorek analyzovaný v rámci předchozí kapitoly je upravená verze malware známého pod názvem Cridex (úprava spočívá pouze v bezvýznamné změně vedoucí ke změně kryptografické haše se záměrem vyhnout se detekci pomocí API VirusTotal, z důvodu ztížení analýzy malware v této ukázkové situaci).

Ačkoliv není žádané v případě nalezení vzorku nahrát tento vzorek do nástroje VirusTotal k provedení analýzy (což by v předchozí ukázkové situaci bylo nutné, protože hash vzorku není v tomto případě nástroji VirusTotal známa), podívejme se nejdříve na rozdíly s výstupem analýzy tohoto malware nástrojem VirusTotal jakožto porovnatelného nástroje jež je veřejně dostupný.

Výhodou nástroje VirusTotal je jeho komunitní užití, tedy především sekce komentářů pro daný vzorek, skóre tvořené hlasováním komunity a pak širokým užíváním databází názvů pod nimiž byl tento vzorek vložen k analýze. Ovšem jako zdroje tvořené komunitou tyto údaje mohou být zkreslené či přímo nepravdivé, je tedy třeba k nim přistupovat s patřičným nadhledem. Z hlediska statické analýzy nástroj VirusTotal neposkytuje žádné informace navíc, snad pouze adresu vstupního bodu programu, která v technické zprávě

nástroje VirusTotal není poznačena (především protože sama o sobě je tato informace spíše méně podstatná, pro případnou další analýzu lze však tuto informaci nalézt v detailním výpise nástroje ForensIRT, jež je dostupný ve staženém archivu s detaily analýzy nástrojem ForensIRT). Z hlediska dynamické analýzy nástroj VirusTotal také poskytuje seznam registrů k nimž vzorek přistupoval, nejen ty do nichž zapisoval (tato informace byla z nástroje ForensIRT záměrně vynechána, protože počet takových záznamů byl příliš velký a tím klesala jeho informační hodnota a přínos při rychlé analýze, k níž je ForensIRT určen). Navíc výsledky dynamické analýzy nástrojem VirusTotal obsahují více záznamů o zapsaných registrech windows (o 3 více), což může být způsobeno prostředím v němž je vzorek malware analyzován či dobou běhu programu (mohlo dojít k ukončení analýzy ještě předtím, než malware stihl tyto změny provést). Naopak nástroj ForensIRT poskytuje mnoho informací navíc, především funkcionalitu ohledně textových řetězců ve vzorku, jež byly klíčové k předběžnému určení schopností malware. Mezi další informace co ForensIRT poskytuje navíc oproti nástroji VirusTotal patří moduly YARA, dynamická analýze síťového provozu, určité podrobnosti o sekcích formátu PE, snímky obrazovky z běhu programu při dynamické analýze a další. Je tedy zřejmé, že v případě testovaného vzorku ForensIRT poskytuje více informací, které jsou zároveň relevantní a dobře srozumitelné.

VirusTotal patří mezi nejpopulárnější nástroje využívané k vyhodnocení škodlivosti vzorku software. Na druhou stranu Cuckoo patří mezi nejpopulárnější otevřené nástroje jež si každý může vytvořit a může tím zamezit sdílení informací o analyzovaných vzorcích s třetími stranami. Avšak protože nástroj ForensIRT vychází z nástroje Cuckoo a používá jej k dynamické analýze, nástroj ForensIRT poskytuje všechny informace poskytované tímto nástrojem. Navíc je však obohacen o velké množství informací získané ze statické analýzy, především podrobnosti o formátu PE, či schopnosti analýzy signaturami YARA. Tedy i v tomto případě nástroj ForensIRT poskytuje více relevantních informací než nástroj Cuckoo.

Kapitola 8

Závěr

V této diplomové práci byly probrány aktuální trendy v metodice a postupech při statické a dynamické analýze malware. Byly prozkoumány existující nástroje využívané při analýze malware a byly probrány jejich možnosti a přínosy při analýze malware, ať už se jednalo o jednostranné specializované nástroje, či balíky nástrojů schopné samostatně provést rozsáhlé analyzační úkony. Bylo zjištěno že existují mocné nástroje, avšak některé z nich – jako například nejznámější nástroj VirusTotal – jsou nevhodné pro použití bezpečnostními týmy, a to především proto, že sdílí informace o analyzovaném malware s veřejnou komunitou uživatelů tohoto nástroje. Následně byly diskutovány postupy bezpečnostních týmů, jejich cílů a specifických potřeb při analýze malware. Zátěž bezpečnostních týmu je již nyní vysoká. Bezpečnostní týmy jsou omezeny jak finančně, tak i počtem schopných a vysoce odborně vzdělaných pracovníků, především forenzních specialistů. Navíc často poprvé do styku s novými vzorky malware přichází mladší a méně zkušení členové týmu v rámci řešení běžných bezpečnostního incidentu. Proto nástroj pro forenzní analýzu malware bezpečnostními týmy musí poskytovat přehledné a kompletní výsledky, avšak prezentované ve formě, která umožňuje porozumění právě těmto méně zkušeným členům týmu, jež budou tento nástroj využívat nejčastěji.

Na základě těchto a dalších potřeb byl navrhnut modulární nástroj pro analýzu malware s názvem ForensIRT. Následně byl tento nástroj implementován v jazyce Python za využití technologií Django, Cuckoo a dalších specializovaných knihoven. Při vývoji byl kladen důraz na poskytnutí přehledné avšak kompletní technické zprávy, jež obsahuje relevantní informace předané srozumitelným způsobem uživatelům jež mají jen málo zkušeností s forenzní analýzou malware, jež jsou typickým uživatelem tohoto nástroje v rámci řešení incidentů zachycených těmito bezpečnostními týmy. Robustní infrastruktura nástroje a flexibilní modulární systém navíc umožňují snadnou údržbu a budoucí vývoj nových modulů implementujících nové metody a postupy vycházející z budoucích trendů ve forenzní analýze malware.

Konečně na závěr této práce byla provedena analýza vzorku malware Cridex, na kterém byla demonstrována vysoká schopnost a použitelnost vyvinutého nástroje ForensIRT, která byla zakončena porovnáním dosažených výsledků s nástroji VirusTotal a Cuckoo, jakožto zástupci moderních nejpoužívanějších nástrojů pro forenzní analýzu malware. Na základě dosažených výsledků je zřejmé, že nástroj ForensIRT je velmi vhodný pro použití bezpečnostní týmy, pro jejichž potřeby byl cíleně navržen. Nástroj ForensIRT v tomto testu implementuje všechny metody jako porovnávané nástroje VirusTotal a Cuckoo, ale navíc poskytuje i další důležité informace nad rámec těchto nástrojů.

Nástroj ForensIRT má navíc velký potenciál v budoucím vývoji. Jelikož infrastruktura nástroje je velmi robustní a počítá s budoucím vývojem dalších analyzačních modulů, jako další možné rozšíření se nabízí právě vývoj dalších takových modulů implementujícím například nové statické či dynamické analyzační metody. Příkladem takových nových modulů může být podpora více metod a postupů, jako například podpora více formátů jejichž součástí může být malware – například analýza infikovaných dokumentů formátu Word a Excel. Dalším možným směrem vývoje by mohlo být vytvoření databáze praivdel YARA obsahující signatury známých vzorků malware, což by vedlo ke schopnosti detekovat nástrojem ForensIRT nejznámější vzorky malware nezávisle na výsledcích modulu VirusTotal. Samozřejmě možným vylepšením by mohla být také implementace metod využívajících strojové učení a umělou inteligenci ke klasifikaci nových vzorků malware na základě artefaktů a vzorů často viděných u již známých vzorků malware.

Popřípadě by bylo zajímavé v budoucím vývoji vybudovat systém skórování analyzovaných vzorků jednotlivými moduly, jež by mohlo vést k automatickému skórování analyzovaných vzorků a tedy možnosti využití nástroje k hromadné automatizované analýze velkého objemu vzorků malware, což by mohlo vést k vytlačení potřeby lidského operátora a což by vedlo ke zvýšení možného mnoství analyzovaných vzorků malware bez nutnosti navýšení lidských zdrojů potřebných k této analýze. Alternativně by toto navrhované automatické skórování mohlo být použito k předběžnému automatickému filtrování zřejmě neškodlivých vzorků a tak lidskému operátorovi nástroje poskytnout k ruční analýze jen zajímavé a mnohem spíše infikované vzorky, což by dále vedlo ke snížení pracovní zátěže bezpečnostních týmů, jakožto hlavního problému jež je třeba v této oblasti řešit.

Literatura

[1] Berry, A.; Homan, J.; Eitzman, R.: WannaCry. Malware Profile, FireEye, May 2017 [cit. 2018-05-01].
Dostupné z: https://www.fireeye.com/blog/threatresearch/2017/05/wannacry-malware-profile.html

[2] Choi, Y. S.; Kim, I. K.; Oh, J. T.; aj.: PE File Header Analysis-Based Packed PE File Detection Technique (PHAD). In *International Symposium on Computer Science and its Applications*, Oct 2008, ISSN 2159-7030, s. 28–31, doi:10.1109/CSA.2008.28.

[3] Christodorescu, M.; Jha, S.; Seshia, S. A.; aj.: Semantics-aware malware detection. In *2005 IEEE Symposium on Security and Privacy (S P'05)*, May 2005, ISSN 1081-6011, s. 32–46, doi:10.1109/SP.2005.20.

[4] CSIRT.CZ: Zpráva o činnosti CSIRT.cz (Národního CSIRT ČR) za rok 2016. In *Výroční zpráva*, July 2016 [cit. 2018-05-01], s. 1–13.
Dostupné z:
https://csirt.cz/files/csirt/Zprava_o_cinnosti_CSIRT.CZ.2016.pdf

[5] Garnaeva, M.; Sinitsyn, F.; Namestnikov, Y.; aj.: Overall Statistics for 2016. In *Kaspersky Security Bulletin*, Oct 2016 [cit. 2018-05-01], s. 1–31.
Dostupné z: https://kasperskycontenthub.com/securelist/files/2016/12/Kaspersky_Security_Bulletin_2016_Statistics_ENG.pdf

[6] Inoue, D.; Yoshioka, K.; Eto, M.; aj.: Malware Behavior Analysis in Isolated Miniature Network for Revealing Malware's Network Activity. In *2008 IEEE International Conference on Communications*, May 2008, ISSN 1550-3607, s. 1715–1721, doi:10.1109/ICC.2008.330.

[7] Lyda, R.; Hamrock, J.: Using Entropy Analysis to Find Encrypted and Packed Malware. *IEEE Security Privacy*, ročník 5, č. 2, March 2007: s. 40–45, ISSN 1540-7993, doi:10.1109/MSP.2007.48.

[8] McAnlis, C.; Haecky, A.: *Understanding Compression: Data Compression for Modern Developers*. O'Reilly Media, Inc., první vydání, 2016, ISBN 1491961538, 9781491961537.

[9] Microsoft: PE Format. [cit. 2018-05-01].
Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ms680547(v=vs.85).aspx

[10] Mockapetris, P.: Domain Names - Concepts and Facilities. RFC 1034, The Internet Engineering Task Force, November 1987.
Dostupné z: https://www.ietf.org/rfc/rfc1034.txt

[11] Morales, J. A.; Al-Bataineh, A.; Xu, S.; aj.: *Analyzing and Exploiting Network Behaviors of Malware*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ISBN 978-3-642-16161-2, s. 20–34, doi:10.1007/978-3-642-16161-2_2.
Dostupné z: https://doi.org/10.1007/978-3-642-16161-2_2

[12] Oikarinen, J.; Reed, D.: Internet Relay Chat Protocol. RFC 1459, The Internet Engineering Task Force, May 1993.
Dostupné z: https://www.ietf.org/rfc/rfc1459.txt

[13] PAČKA, R.: Role národních Computer Emergency Response Teams (CERT) v zajišťování kybernetické bezpečnosti státem [online]. 2017 [cit. 2018-05-01].
Dostupné z: http://is.muni.cz/th/333252/fss_r/

[14] Penedo, D.: Technical Infrastructure of a CSIRT. In *International Conference on Internet Surveillance and Protection (ICISP 146;06)*, Aug 2006, s. 27–27, doi:10.1109/ICISP.2006.32.

[15] Rieck, K.; Holz, T.; Willems, C.; aj.: Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2008, s. 108–125.

[16] Ruefle, R.; Dorofee, A.; Mundie, D.; aj.: Computer Security Incident Response Team Development and Evolution. *IEEE Security Privacy*, ročník 12, č. 5, Sept 2014: s. 16–26, ISSN 1540-7993, doi:10.1109/MSP.2014.89.

[17] Sikorski, M.; Honig, A.: *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. San Francisco, CA, USA: No Starch Press, první vydání, 2012, ISBN 1593272901, 9781593272906.

[18] Tian, R.; Batten, L.; Islam, M.; aj.: *An automated classification system based on the strings of trojan and virus families*. Institute of Electrical and Electronics Engineers, 2009, s. 23–30, doi:10.1109/MALWARE.2009.5403021.

[19] Uppal, D.; Sinha, R.; Mehra, V.; aj.: Malware detection and classification based on extraction of API sequences. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sept 2014, s. 2337–2342, doi:10.1109/ICACCI.2014.6968547.

[20] Wang, T.-Y.; Wu, C.-H.; Hsieh, C.-C.: Detecting Unknown Malicious Executables Using Portable Executable Headers. In *NCM 2009 - 5th International Joint Conference on INC, IMS, and IDC*, 01 2009, s. 278–284.

[21] Wang, Y.; Gu, D.; Peng, D.; aj.: *Stuxnet Vulnerabilities Analysis of SCADA Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-35211-9, s. 640–646, doi:10.1007/978-3-642-35211-9_81.
Dostupné z: https://doi.org/10.1007/978-3-642-35211-9_81

[22] Wiik, J.; Gonzalez, J. J.; Davidsen, P. I.; aj.: Chronic Workload Problems in CSIRTs. July 2009, doi:10.1.1.640.7783.

[23] Wojner, C.: Mass Malware Analysis: A Do-It-Yourself Kit. October 2009 [cit. 2018-05-01].
Dostupné z:
https://cert.at/downloads/papers/mass_malware_analysis_en.html

[24] Yannakogeorgos, P. A.; Tikk, E.: Stuxnet as cyber-enabled sanctions enforcement. In *2016 International Conference on Cyber Conflict (CyCon U.S.)*, Oct 2016, s. 1–6, doi:10.1109/CYCONUS.2016.7836630.

[25] Zhang, Q.; Reeves, D. S.: MetaAware: Identifying Metamorphic Malware. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, Dec 2007, ISSN 1063-9527, s. 411–420, doi:10.1109/ACSAC.2007.9.

Příloha A

Analýza ransomware Wannacry nástrojem ForensIRT

V této příloze je provedena analýza vzorku malware známého pod názvem WannaCry (nechvalně známý vzorek malware, jež byl poprvé spatřen v roce 2017 a napadl desetitisíce až statisíce počítačů po celém světě), aby byla dále demonstrována funkčnost nástroje ForensIRT a jeho schopnost zpracovat a úspěšně analyzovat i aktuální moderní vzorky malware.

Předmětem analýzy je vzorek s následujícími parametry:

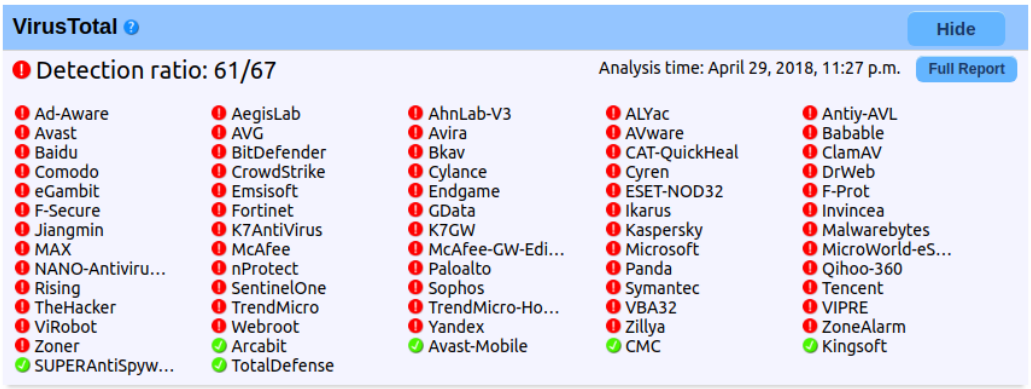
- Název souboru: `diskpart.exe`
- Velikost souboru: `3 514 368 Bajtů`
- MD5 Hash: `84c82835a5d21bbcf75a61706d8ab549`
- SHA-256 Hash: `ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6 e5babe8e080e41aa`

Již při pohledu na první analyzační modul – VirusTotal – je zřejmé, že analyzovaný vzorek je malware, protože 61 ze 67 antivirů spuštěných službou VirusTotal identifikovalo tento vzorek jako škodlivý. Po bližším přezkoumání pojmenování jež tyto antiviry přidělily tomuto vzorku se zřejmě jedná o malware s názvem Wannacry či Wannacryptor z rodiny ransomware. I přesto však z důvodů demonstrace analýzy malware nástrojem ForensIRT pokračujme v analýze dalšími moduly (pak také z toho důvodu, že snadným pozměněním jediného bitu ve vzorku by byl nástrojem VirusTotal nedetekován, což by mohl útočník využít při případném útoku).

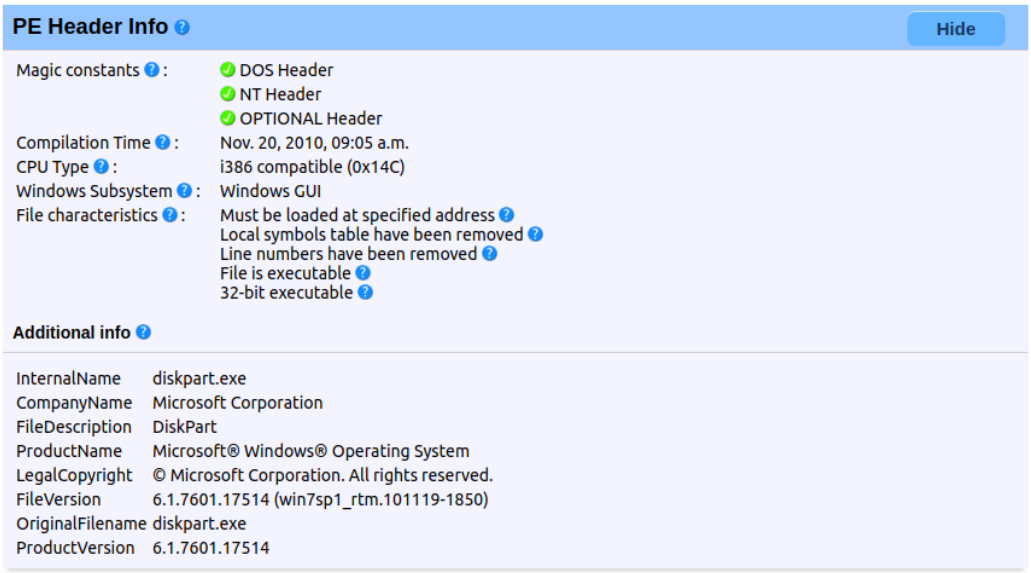
Následující analyzační modul PE Header Info (viz obrázek **??**) poskytuje základní informace získané ze záhlaví zkoumaného vzorku. Zřejmě se jedná o 32bit aplikaci určenou pro MS Windows s podporou pro grafické uživatelské rozhraní. Navíc je v záhlaví přiložena série dvojic řetězců jež informují že se jedná o program DiskPart od společnosti Microsoft. Ovšem tato data mohou být podvržená, především protože soubor není podepsán klíčem společnosti Microsoft. Proto je již okažitě jasné, že zcela určitě jde o nějaký škodlivý program jež se vydává za součást operačního systému Windows – konkrétně program `DiskPart.exe`.

Modul PE Import Directory (viz obrázek A.3) informuje uživatele že analyzovaný vzorek používá celkem 4 knihovny DLL ze kterých importuje především funkce pro práci se soubory a s operačním systémem (včetně funkcí pro maniupulaci se systémovými registry, spouštění systémových procesů a služeb a dalších). Ovšem jedna z importovaných funkcí

Obrázek A.1: Výsledky analýzy modulu Virus Total



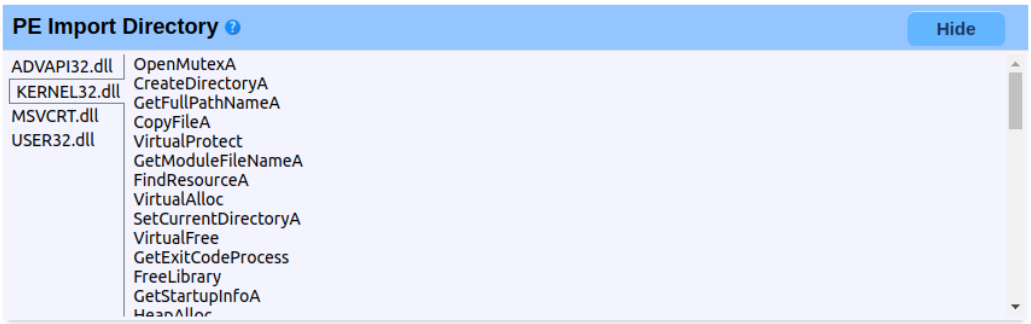
Obrázek A.2: Výsledky analýzy modulu PE Header Info



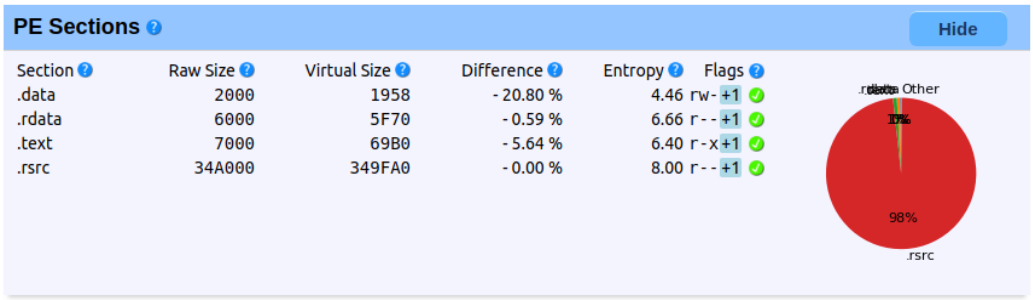
(`LoadLibraryA`) umožňuje za běhu načítat další knihovny a importovat z nich funkce, takže nelze ze statické analýzy spoléhat na zjištěné výsledky, protože při běhu může tento vzorek spoouštět další proces, resp. importovat další funkce z dalších knihoven, nelze tedy s jistotou nyní určit jeho schopnosti. Navíc je importována funkce `CryptReleaseContext` jež slouží k práci s kryptografickými klíči. Zřejmě tedy bude tento vzorek provádět nějaké kryptografické úkony.

Z následujícího analyzačního modulu PE Sections (viz obrázek A.4) lze vyčíst, že naprostá většina dat programu je obsažena v sekci `.rsrc` a tato sekce má maximální možnou entropii (8 z 8), zřejmě tedy půjde o šifrovaná data. Žádná ze sekcí nemá příznaky zápisu i spuštění (`write` a `execute`), takže vzorek nemůže za běhu měnit svou funkcionalitu, ovšem protože importuje funkci pro spuštění nového procesu a funkce pro práci se soubory, nic nezabraňuje vzorku vytvořit nový soubor do kterého rozšifruje obsah sekce `.rsrc` a poté jej spustí. Může se tedy jednat o program jež pouze rozbalí samotný škodlivý kód, neboli tzv. `dropper`.

Obrázek A.3: Výsledky analýzy modulu PE Import Directory

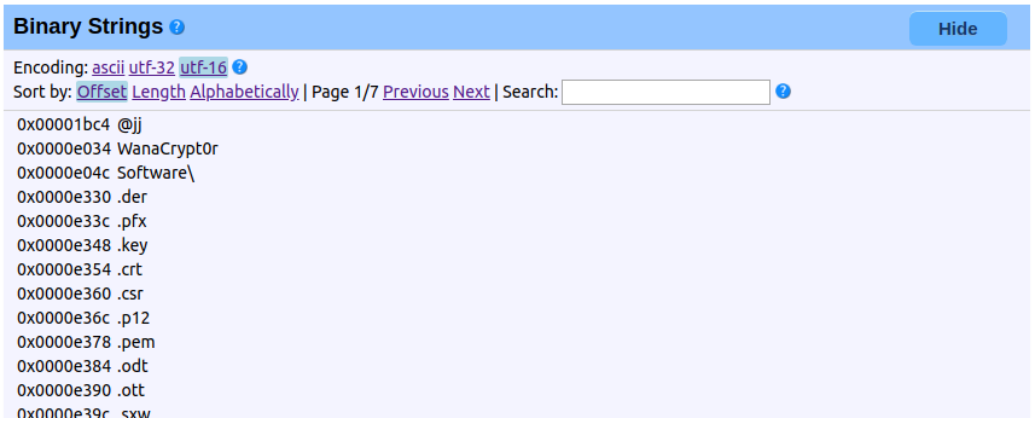


Obrázek A.4: Výsledky analýzy modulu PE Sections



Z následujícího analyzačního modulu Binary Strings (viz obrázek A.5) lze vyčíst, že program neobsahuje žádné běžné řetězce, jako například chybové hlášky či řetězce umožňující práci s programem. Naopak obsahuje mnoho řetězců v kódování `UTF-16` jež jsou zřejmě koncovky souborů. Navíc je zde k nalezení řetězec `WanaCrypt0r`, z nějž také vychází název tohoto malware. Seznam koncovek nalezený modulem Strings tedy zřejmě bude seznam koncovek jež tento malware zašifruje.

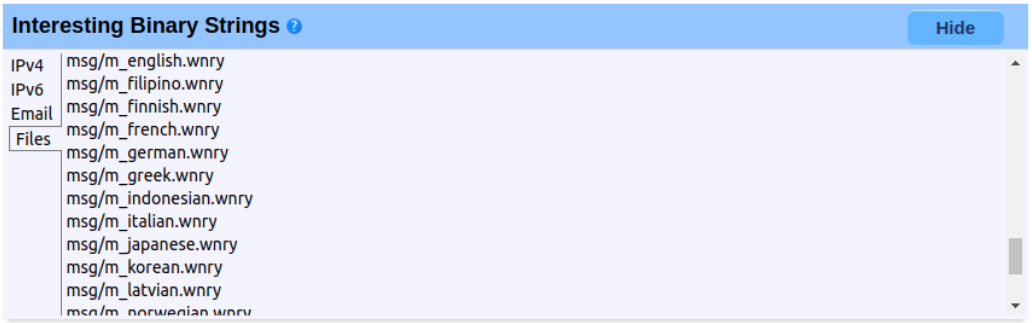
Obrázek A.5: Koncovky souborů jež budou malware WannaCry zašifrovány



Modul Interesting Binary Strings (viz obrázek A.6) dále vypichuje některé zajímavé řetězce. Především je zde zřejmá série řetězců jež jsou relativní cestou k souborům s názvem některého světového jazyku s prefixem `m_` a koncovkou `.wnry` ve složce `msg`. Pravděpodobně se jedná o soubory kam budou rozšifrovány řetězce v mnoha světových jazycích, aby byl

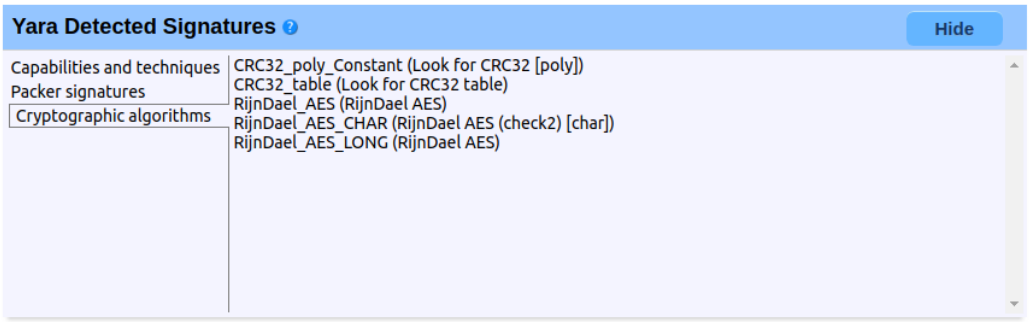
spuštěný vzorek použitelný (tedy jedná se o soubory kam budou nahrány řetězce vyzývající uživatele k zaplacení určité částky aby byly jejich soubory rozšifrovány).

Obrázek A.6: Výsledky analýzy modulu Interesting Binary Strings



Modul Yara Detected Signatures (viz obrázek A.7) identifikoval přítomnost několika různých šifrovacích algoritmů, především algoritmu `Armadillo`, jímž budou pravděpodobně zašifrována data v sekci `.rsrc`. Navíc Byly v kódu nalezeny signatury odpovídající kryptografickým funkcím AES a CRC32. Toto zjištění potvrzuje dřívější podezření na schopnost programu šifrovat a dešifrovat data pomocí asymetrické kryptografie.

Obrázek A.7: Výsledky analýzy modulu Yara Detected Signatures

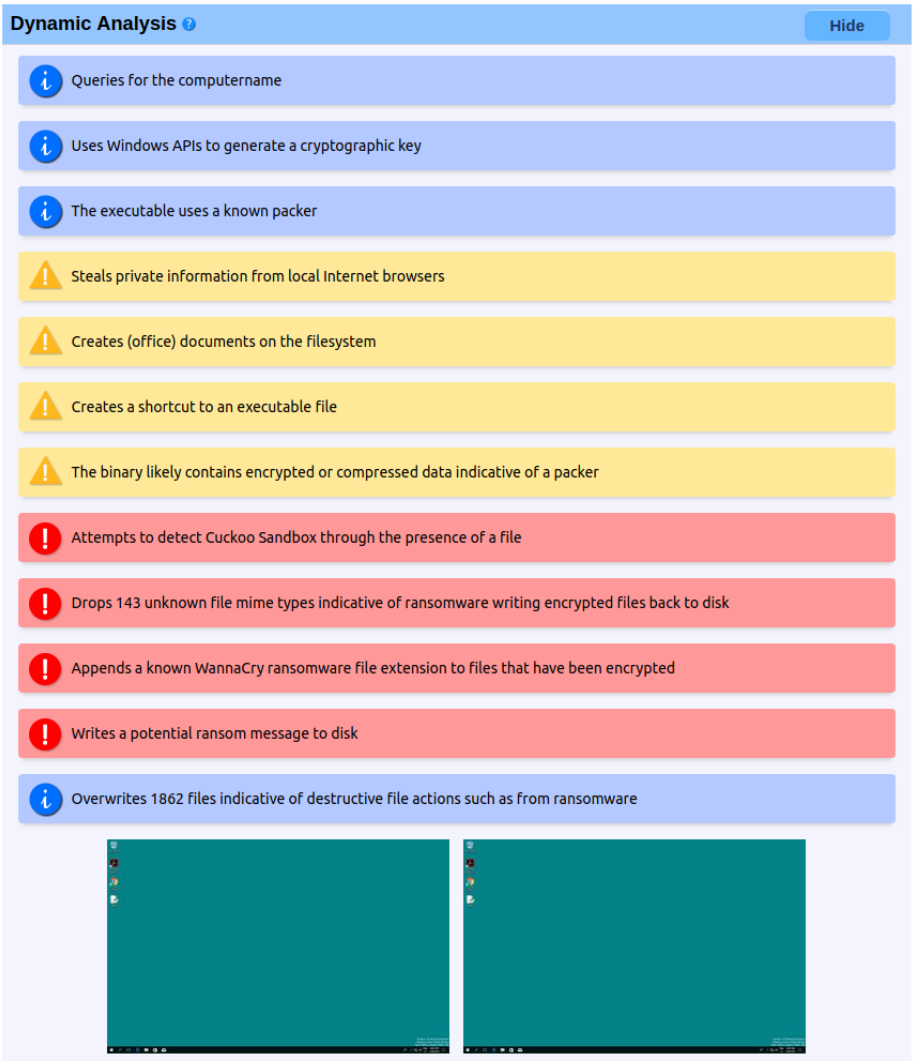


Modul Dynamic Analysis (viz obrázek A.8) informuje uživatele že vzorek skutečně obsahuje šifrovaná data (pravděpodobně se jedná o data v sekci `.rsrc`) a skutečně program užívá Windows API k vytvoření kryptografických klíčů. Navíc modul varuje že vzorek je schopen (a takto skutečně činí) krást data z webového prohlížeče. Navíc se vzorek pokusil zjistit zda se nachází ve virtuálním analyzačním prostředí. Konečně modul také zobrazuje varování že v době analýzy program stihl zapsat 143 neznámých souborů na disk, ke kterým připojil koncovku známou jako koncovku šifrovaných programů malwarem WannaCry a na disk byl zapsán soubor obsahující potenciální zprávu o zaplacení výkupného pro dešifrování souborů (pravděpodobě se jedná o soubory `msg/m_<lang>.wnry`). Navíc vzorek po dobu svého běhu přepsal obsah 1862 souborů (než byl jeho běh ukončen na konci analýzy). Tyto skutečnosti jasně indikují že se jedná o malware s názvem WannaCry.

Výsledky modulu Dynamic Analysis – Network Traffic (viz obrázek A.9) nepřináší žádné nové informace. V době programu byly vyslány dotazy na dostupnost domén s běžnými názvy. Nebyla zachycena žádná podezřelá síťová činnost vzorku malware.

Naopak výsledky modulu Dynamic Analysis – Sample Behavior (viz obrázek A.10) přináší podrobnosti o již dříve zjištěných schopnostech a činnostech vzorku. Konkrétně že

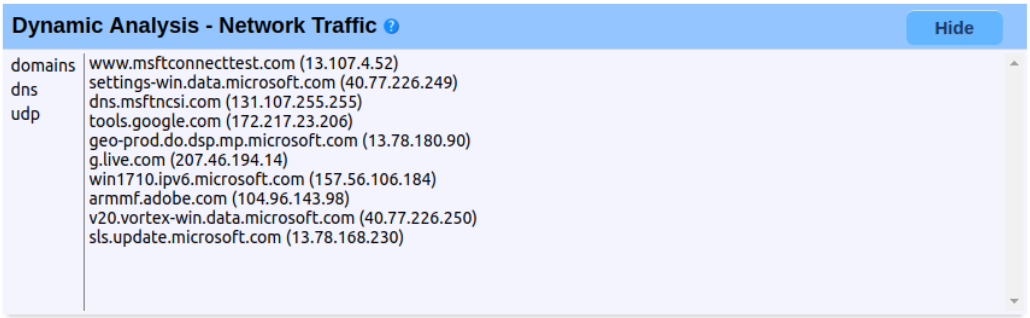
Obrázek A.8: Výsledky analýzy modulu Dynamic Analysis



byla vytvořena složka `C:\Users\user\AppData\Local\Temp\msg`, kam byly zřejmě dekódovány soubory řetězci informující uživatele o nutnosti vyplatit výkupné aby jeho data byla dešifrována, jak již bylo zmíněno výše. Dále je zřejmé že vzorek malware upravoval systémové registry a navíc pracoval s obrovským množstvím souborů, jež různě upravoval, přejmenovával apod.

Z analýzy tohoto vzorku je bezpochyby zřejmé že se jedná o nechvalně známý ransomware WannaCry. Tuto skutečnost potvrzuje nezávisle na sobě několik modulů založených jak na statické tak i dynamické analýze vzorku. Navíc nástroj ForensIRT přímo umožnil vyšetřovateli pochopit základní body funkčnosti vzorku a identifikovat použité šifrovací algoritmy.

Obrázek A.9: Výsledky analýzy modulu Dynamic Analysis - Network Traffic



Obrázek A.10: Výsledky analýzy modulu Dynamic Analysis - Sample Behavior